LeonardoSpectrum User's Guide

v1999.1

Copyright

Copyright © 1991-1999 Exemplar Logic, Inc., A Mentor Graphics Company

All Rights Reserved

Trademarks

Exemplar Logic™ and its Logo are trademarks of Exemplar Logic, Inc.
LeonardoSpectrum™, LeonardoInsight™, FlowTabs™, HdlInventor™, SmartScripts™, P&RIntegrator™, DesktopASIC™, XlibCreator™, SynthesisWizard™, and MODGEN™ are trademarks of Exemplar Logic, Inc.; Model Sim/VHDL™, Model Sim™, and V-System/Verilog™ are trademarks of Model Technology, Inc.; Renoir™, Monet™, and PackagedPower™ are trademarks of Mentor Graphics Corporation. Verilog® and Verilog-XL® are registered trademarks of Cadence Design Systems, Inc.
All other trademarks remain the property of their respective owners.

Disclaimer

Although Exemplar Logic, Inc. has tested the software and reviewed the documentation, Exemplar Logic, Inc. makes no warranty or representation, either express or implied, with respect to this software and documentation, its quality, performance, merchantability, or fitness for a particular purpose.

Contents

1. Introduction 1-1

HDL Solution 1-1

LeonardoSpectrum Modular Levels 1-2

Level 1 1-2

Level 2 1-3

Level 3 1-4

Options 1-4

Level 3 - DesktopASIC 1-5

All Three Levels 1-5

LeonardoInsight 1-5

HDL Languages 1-6

Standard Features 1-6

Save and Restore Project 1-6

P&RIntegrator 1-7

HDLInventor 1-7

Design Browser 1-8

Still More Features 1-8

This LeonardoSpectrum User's Manual 1-8

Available Online and Website 1-8

Available Online Context-Sensitive Help 1-8

Available Libraries 1-9

Tcl Script Sourcing 1-9

XlibCreator for ASIC - Complete Development Kit 1-9

Screen Shots, Reports, Filenames, and Code Examples 1-10

Three Ways to Synthesis 1-10
SynthesisWizard 1-10
Quick Setup 1-10
FlowTabs 1-10
Conventions Used 1-11
PC Hardware and Software Requirements 1-11
PackagedPower 1-12
2. Graphical User Interface 2-1
Startup 2-1
Tip of the Day 2-2
SynthesisWizard 2-3
Main Window Description 2-3
Main Window Header 2-4
Information Window 2-12
Command Line Window 2-12
HDLInventor 2-13
Status Bar 2-19
3. Level 2 FlowTabs 3-1
Synthesis FlowTabs 3-1
Quick Setup Tab Active Review - FPGA 3-2
Technology Tab - FPGA 3-4
Input Tab 3-6
Constraints Tab 3-8
Optimize Tab 3-10
Output Tab 3-12
P&R Tab 3-14
Back Annotation Tab 3-20
4. Level 3 FlowTabs 4-1

Synthesis FlowTabs 4-1
Quick Setup Tab - Active Review - FPGA 4-2
Quick Setup Tab - Active Review - ASIC. 4-4
Technology Tab - FPGA 4-6
Advanced Settings. 4-6
Technology Tab - ASIC 4-8
Input Tab. 4-10
Constraints Tab 4-12
Optimize Tab. 4-14
Report Tab. 4-16
Output Tab 4-20
P&R Tab 4-22
Back Annotation Tab 4-28
Additional Instructions 4-30
Synthesis Steps for Output Netlist 4-30
Retarget Steps for Output Netlist 4-30
Steps for auto_dissolve 4-31
HP Platform: Out-of-Memory Workaround 4-32
Back Annotation Neoprim and Simprim Hints 4-33
APEX 20K/20KE Variables and Other APEX 20K/20KE Information. 4-33
Mapping Options. 4-33
APEX 20K/20KE Mapping. 4-33
More Quartus Information. 4-35
Simpler Clock Enable Logic 4-35
5. Constraints 5-1
Constraint File 5-1
Syntax. 5-2
Constraint File Commands 5-2

Syntax for Busses 5-2
Alphabetical List 5-3
Attributes 5-6
Load and Drive Specifications 5-6
Preserving Signals 5-7
Buffering Specifications 5-9
Timing Requirements 5-10
Input/Output Timing Parameters 5-11
Register Timing 5-11
HDL Examples 5-13
Command Examples 5-13
Low Level Block Constraints 5-14
Constraint Editor 5-15
Special Steps for Quick Setup - Level 2 5-31
6. Batch Mode Options 6-1
Alphabetical List 6-1
Options for MAX+PLUS II Functionality 6-23
License Information 6-23
Tcl Script Sourcing 6-24
Interactive Command Line Shell (Level 3) 6-24
GUI Menu Bar File -> Run Script 6-24
Command Line with Path to LeonardoSpectrum 6-24
7. Power Tabs and Advanced Topics 7-1
Input File Options 7-1
Output File Options 7-7
Optimize Options 7-11
Advanced Technology FPGA 7-14
Advanced Technology ASIC 7-16

Special Instructions for Mixing Design Languages 7-17
Adding a Library to the GUI (Level 3) 7-19
Adding a Library to the GUI 7-20
Special Instructions for Adding Libraries 7-21
8. LeonardoInsight 8-1
Cross Probing 8-2
Cross Probing Between Renoir and LeonardoSpectrum 8-2
HDL Constructs that Initiate Cross Probing 8-3
Cross Probing from HDL Source Code to Schematic Viewer 8-4
Cross Probing to HDL Source Code 8-7
Cross Probing Between RTL and Gate Level Schematic 8-8
Schematic Fragments 8-8
Schematic Fragments for Critical Path Analysis Example 8-8
Schematic Cones for Fanin or Fanout Logic from Selected Nodes 8-11
Schematic Viewer Description 8-16
Bring Up Schematic Viewer 8-16
How to Use the Schematic Viewer 8-17
Menu Bar 8-17
Tool Bar 8-20
RMB Popup Menu 8-22
Strokes 8-23
Design Browser Description 8-23
Browser Windows 8-24
Library Window 8-24
Hierarchy Window 8-25
Adding a Technology Specific Symbol Library 8-28
9. Design Database for Level 3 9-1
Design Data Information Model 9-1

Accessing Design Data 9-2

10. Application Notes 10-1

Pipelined Multiplier 10-1

Introduction 10-1

The LeonardoSpectrum Approach to Pipelining 10-1

Variable 10-2

Quality of Results 10-2

Note: Clock Enable, Asynchronous Clear/Set 10-3

Example Template VHDL and Verilog RTL 10-3

Altera APEX 20K/20KE PTERM (Product-Term) Support: 10-6

Usage: 10-6

Save and Restore Project 10-7

What is a Project? 10-7

Advantages of Using Project 10-7

Starting a Project 10-8

Tips and Tricks - Examples: 10-9

11. Menu Bar Items 11-1

Session Settings 11-1

Browser Filter 11-8

Variable Editor 11-9

12. Design Methodology 12-1

ASIC Flow Charts 12-2

Example Flow Session 12-5

Environment Setup 12-9

Design Partitioning 12-12

Synthesizing Designs 12-15

Adding Cells and Modeling Memories 12-22

Setting Constraints 12-28

Global Optimization Variables 12-45

Clock Buffering 12-53
Optimization 12-55
Hold Time Analysis and Netlist Connection 12-63
Saving Results 12-65
Back Annotated Static Timing Analysis and Optimization 12-66
Referencing Netlist Objects 12-67
Hierarchy Manipulation 12-67
LeonardoSpectrum and Design Compiler Commands 12-70
13. SynthesisWizard Tutorial 13-1
SynthesisWizard Tour 13-1
Device Settings - Step 1 of 4 13-2
Input Files, Step 2 of 4 13-3
Global Constraints, Step 3 of 4 13-6
Output File, Step 4 of 4 13-7
Run 13-8
14. Messages 14-1
Information Message 14-1
Error Message 14-1
Warning Message 14-2

Introduction 1

Welcome to LeonardoSpectrum. LeonardoSpectrum is a suite of high level design tools for a Complex Programmable Logic Device (CPLD), Field Programmable Gate Array (FPGA), or Application Specific Integrated Circuit (ASIC). LeonardoSpectrum offers design capture, VHDL and Verilog entry, register transfer level debugging for logic synthesis, constraint based optimization, timing analysis, encapsulated place-and-route, and schematic viewing. This introduction is divided as follows:

- HDL Solution
- LeonardoSpectrum Modular Levels
- Options
- Standard Features
- Still More Features
- Three Ways to Synthesis
- Conventions Used
- PC Hardware and Software Requirements
- PackagedPower

HDL Solution

A complete hardware description language (HDL) solution is here for Windows 95/98/NT; and UNIX HP and Sun. A quality design is completed for you with each of the LeonardoSpectrum tools: Level 1, Level 2, and Level 3. A native Windows graphical user interface (GUI) is common to all three levels and provides these features:

- Same look and feel for all levels
- Windows editing, dragging, and dropping attributes are available
- SynthesisWizard, Quick Setup, and FlowTabs guide you through the design process
- Embedded, interactive, and filtered windows extend task information
- Quick file changes with right mouse button (RMB)
- Popups and pulldowns are prevalent
- Pertinent information is parsed for quick reading
- Clickable buttons assign tasks

LeonardoSpectrum Modular Levels

This section compares and contrasts the three modular tool levels. All three levels are powered by the LeonardoSpectrum core synthesis and optimization engine which yields superior design results with a minimum of tool manipulation; and at the same time, allows you to control the design domain.

As described in the following paragraphs, the design methodology becomes more detailed with each successive level: Level 1 produces the basic netlist, Level 2 adds more intricate design capabilities; and Level 3 contributes the ultimate in interactive, advanced features.

Level 1

Level 1 is an easy-to-use, single FPGA technology, synthesis tool that uses the LeonardoSpectrum database. A logic designer selects the input design and Xilinx technology (for example) and then clicks the Run button. A high-quality netlist is quickly produced. Level 1 includes the following clearly defined features:

- Windows 95/98/NT node-locked platform
- Single FPGA vendor
- Certified FPGA flows
- Global constraints (frequency, period, input to register, register to register, register to output, input to output)
- Technology-independent specifications
- Technology-specific operator generation
- Architecture-specific optimization
- Place and route sub-invocation (P&RIntegrator) for Xilinx, Altera, Altera Quartus, and Lattice/Vantis technologies.

- Integrated source code editor (HDLInventor) including template insertion and error/warning cross highlighting
- Single pass area and timing optimization
- Familiar Batch Mode
- TCL Scripting
- Easy upgrade path to Levels 2 and 3

Level 2

Level 2 is an easy-to-use FPGA synthesis and timing analysis device with back-annotation for all FPGA technologies. A logic designer selects the input design and technology and clicks the Run button. A high-quality netlist is quickly produced. In contrast to Level 1, Level 2 is for all FPGA technologies. Level 2 contributes to the extensive features list of Level 1 with the following:

- Platform independent on Windows 95/98/NT or UNIX
- All FPGA technologies
- Certified FPGA flows: Generated netlists and directives successfully pass through the back-end tools; post place-and-route timing information can then be back-annotated for timing analysis, logic verification, and technology retargetting purposes.
- Hierarchy Preservation
- Advanced Constraints
- Advanced Optimization Switches
- Retarget Output Netlist
- Accurate architecture specific timing analysis
- Optimizes designs for area and speed, and accepts designs as either HDL structural netlists or as RTL (register transfer level)
- Vendor specific netlists are produced together with design reports that provide estimates of design performance
- Save and Restore Project
- Back annotated timing analysis from post place-and-route netlist
- Easy upgrade path to Level 3

Level 3

Level 3 is easy-to-use and is a versatile and interactive logic synthesis, optimization, and analysis tool. Level 3 allows the use of technology-independent design methods for FPGA and CPLD devices, and in contrast to Levels 1 and 2, Level 3 optionally supports advanced algorithms to target ASIC technologies. You can perform bottom-up design assembly with technology-mapped netlist. Hierarchy can be preserved, flattened, merged and dissolved. Plus, complex scripts can be written and run through an interactive batch mode operation. The design effort can be accomplished either by an individual engineer or by a team of engineers.

Level 3 utilizes the most powerful state-of-the-art optimization technology to guarantee high-quality results for any FPGA or ASIC technology. Level 3 adds to the long list of Level 2 features with the following:

- Optional ASIC specific module generation, optimization algorithms, design rule resolving, and technology mapping
- Two-way retarget path exists between FPGA synthesis and optional ASIC synthesis
- Mix HDL Design entry - for example, Verilog, VHDL, EDIF
- Interactive Command Line Shell
- Design Browser with Commands
- Advanced TCL Scripting
- Incremental optimization which allows bottom up, top down, and team design
- RTL and gate-level post-synthesis verification

You can run Level 3 from the GUI interactive command line shell or on TCL script files. Batch mode is also available for Level 3. LeonardoSpectrum is designed to give you easy access to the Model Technology Model Sim/QuickHDL simulator.

Level 3 provides a top-down verification flow through VHDL or Verilog with an SDF timing file. LeonardoSpectrum is fully integrated with Model Technology, Inc. (MTI) simulation environment.

Options

Options are available as follows:

- Level 3 - DesktopASIC
- All Three Levels - LeonardoInsight and HDL Languages

Level 3 - DesktopASIC

LeonardoSpectrum, Level 3, is available as ASIC only, or is available as an ASIC option to FPGA. The ASIC library tree is added to the technology browser. The ASIC synthesis flow is improved dramatically to handle high-density designs and to improve run time and to reduce memory consumption. New algorithms are present in the ASIC flow for optimization, mapping and design rule checker (DRC). LeonardoSpectrum uses the same methodology for both FPGAs and ASICs. You can use LeonardoSpectrum to prototype an ASIC using FPGAs, or retarget an FPGA to ASIC or ASIC to FPGA for volume production.

All Three Levels

LeonardoSpectrum has two options for all three levels:

- LeonardoInsight
- HDL Languages (VHDL and Verilog)

LeonardoInsight

LeonardoInsight is here to bring the design database into view. LeonardoInsight includes the design browser and schematic viewer. LeonardoInsight allows you to simplify the complexities of synthesis with an advanced debug and analysis environment.

Design Browser

The design browser displays ports, nets, instances, registers, and primitive cells. Interactive and filtered windows of the design browser are available on the GUI after you read in your design. In summary, the design browser is a graphical representation of the design database. Objects selected and highlighted in the design browser may also be highlighted in the schematic viewer. Furthermore, if the selected object initiates cross probing, then that line of code is highlighted in your HDL source code.

Note: The design browser can be used to apply `dont_touch`, `unmap`,`unfold`,`group`, `ungroup`, and `ungroup all` commands; `group` and `unfold` are only available for **Level 3**.

Note: The design browser is a standard feature for **Level 3**. In addition, the design browser is available to Levels 1 and 2 from the LeonardoInsight option.

Schematic Viewer

The LeonardoSpectrum schematic viewer - based on the latest in rendering algorithms - produces clear and well-organized schematics. The schematic viewer allows you, for example, to: (1) Cross-probe between HDL source code, RTL schematic, and gate-level schematic. This correlation allows for easy debugging. In addition, you can cross probe a schematic generated in Renoir with a schematic generated in LeonardoSpectrum. (2) You can view the whole critical path in one window, even if the path traverses multiple levels of hierarchy. (3) You can view fanout and fanin cones of logic from a selected net or instance. (4) When the critical path viewer is in query mode, detailed timing popup information is displayed for the objects in the critical path. (5) Query mode provides general popup information for every schematic. (6) The schematic viewer search utility allows you to search for instance, net, and port; and lists these items for you in a window. (7) The schematic viewer can cross probe with Renoir.

HDL Languages

By default you are provided with either the Verilog or VHDL language. You can add either Verilog or VHDL as a second language. **Level 3:** refer to Special Instructions for Mixing Design Languages in Chapter 7.

Standard Features

The standard features allow you to complete the entire synthesis task within LeonardoSpectrum. These features are:

- Save and Restore Project (**Levels 2** and **3**)
- P&RIntegrator
- HDLInventor
- Design Browser (**Level 3**)

Save and Restore Project

Entire design projects can now be restored on the same or a different machine. Before you quit a design, you are prompted to save the entire project. Later you can go back and bring up the discontinued project; the restored project is complete with your specifications and windows environment.

Project Saves and Restores:

- File locations for input files, output files, and current working directory
- Database (RTL, gate level) in XDB format (**Level 3**)
- Present design information
- Applied constraints, directives, and attributes
- All tab selection information: source technology, designation technology, file type, hierarchy preservation, global constraints, optimization passes, FSM encoding.

Note: Within the v1999.x series your saved project files are forward and backward compatible, plus you can read v1998.x project files in v1999.x. However, if you want to read v1999.x project files in v1998.x then set the following variable in v1999.x:
`xdb_write_version v1998.x`

P&RIntegrator

P&RIntegrator automatically subinvokes vendor backend place and route tools - Xilinx Alliance Series, Altera MAX+PLUS II, Altera Quartus, Lattice/Vantis - from within LeonardoSpectrum. The vendor's backend tools then create a binary program file which is used to program FPGA devices.

LeonardoSpectrum is the only industry synthesis tool that interfaces directly to selected FPGA and CPLD place and route tools for optimal results. Moreover, LeonardoSpectrum supports back-annotated timing analysis for many vendors. For example, post-routed "simprim" and "neoprim" netlists are generated by the Xilinx Alliance Series environment. Since these libraries are built directly into LeonardoSpectrum, the Xilinx netlists can be read by LeonardoSpectrum. A netlist interface that reads mapped EDIF netlists and SDF back annotation files is also available in LeonardoSpectrum.

HDLInventor

The HDLInventor is an interactive source code editor in LeonardoSpectrum. You can double click on errors, warnings, and information (red, green, and blue dots) in the information window or click on the name of your input file to bring up the HDLInventor. The HDLInventor interactively highlights syntax and synthesis construct errors found during synthesis. You can make your edits in this window and, if required, insert template(s) of HDL code that you frequently use.

Design Browser

The design browser allows you to traverse through the design hierarchy to observe objects like ports, instances, and nets. Refer again to the design browser description in the LeonardoInsight section.

Still More Features

These features are intended to guide you during the synthesis process.

This LeonardoSpectrum User's Manual

The distinctiveness of the LeonardoSpectrum tools and the steps, descriptions, and screen shots in this manual allow you to start designing right away.

Note: This manual assumes that the reader is familiar with the Windows environment and procedures. LeonardoSpectrum may be referred to as **Leonardo** in some special cases: in a table or as a program name. **Level 3** is highlighted to differentiate, where necessary, between Level 2 and Level 3.

Available Online and Website

This manual is available for viewing online with the Adobe Acrobat Reader after LeonardoSpectrum and the Adobe Acrobat Reader are installed from the CD-ROM. **Note:** The online, PDF manual may contain the most recent information. In addition, the manual can be viewed and printed with desktop utilities. The LeonardoSpectrum manuals are available for down loading from the Exemplar website: **http://www.exemplar.com**.

Available Online Context-Sensitive Help

Throughout, LeonardoSpectrum has several avenues of online help: menu bar help, FlowTabs, SynthesisWizard help buttons, and **F1** context-sensitive help. While FlowTabs is active, press **F1** to open a context-sensitive help or press the help button.

Note: The GUI window must be selected first to be in current focus when using **F1**.

Note: F1 does not work on UNIX.

Online help is designed around Window's help properties with the traditional banner, tabs, and buttons. The intent of help is to provide you with synthesis information as quickly as possible. You can continue with your task and get help with context menus at the same time.

Available Libraries

The LeonardoSpectrum license automatically enables all synthesis libraries. Check Exemplar's web site at **http://www.exemplar.com** for more the latest information on ASIC or FPGA libraries. Chapter 7 presents instructions for adding ASIC and FPGA libraries to the GUI. You can add a symbol library to your technology as explained in Chapter 8.

Tcl Script Sourcing

LeonardoSpectrum provides three ways to source your Tcl script. After you create a Tcl script in a standard text editor, you can source your script from LeonardoSpectrum as follows:

- Interactive Command Line Shell (GUI window) (**Level 3**)
- GUI Menu Bar File -> Run Script
- Command Line with Path to LeonardoSpectrum

Refer to Chapter 6, Batch Mode Options, for path syntax.

XlibCreator for ASIC - Complete Development Kit

The XlibCreator provides a library development environment for ASIC. The XlibCreator tools and documentation are available at:

```
ftp://ftp.exemplar.com/outgoing/asiclib_devkit.tar
```

Contact your vendor or Exemplar Logic for the Synopsys .lib library file and for a license.

The XlibCreator is a complete library development kit which contains templates, scripts, and "C" programs designed to create Exemplar synthesis libraries from a Synopsys .lib format. The major XlibCreator software tools consist of:

- Syngen converts Synopsys .lib source to an intermediate library format: Lgen.
- Libgen finishes the process by compiling lgen files into binary .lib synthesis files.

Screen Shots, Reports, Filenames, and Code Examples

The screen shots, reports, filenames, and code examples in this manual may differ slightly from the actual or most current screens and examples. Moreover, some screen shots may have options selected and filenames displayed for illustration purposes only.

Three Ways to Synthesis

LeonardoSpectrum provides three ways to synthesize your design:

- SynthesisWizard
- Quick Setup
- FlowTabs

SynthesisWizard

The SynthesisWizard is designed for the first time user. The SynthesisWizard walks you through the synthesis process. Every step, from specifying a technology to input files to design goals, is clearly presented to you in a SynthesisWizard flow.

Quick Setup

Quick Setup is intended for the user who is familiar with LeonardoSpectrum and the synthesis process. Everything that is specified in the SynthesisWizard, can be specified on one condensed tab. Once specified, you can hit the run button to run the entire synthesis flow including synthesis, global constraints, optimizing and writing netlist. In addition, Quick Setup automatically sets up all options, defaults, and settings in the FlowTabs to assist you when walking through the more advanced tabs.

FlowTabs

The salient FlowTabs are designed for the advanced user who needs access to all the embedded power of LeonardoSpectrum. Nearly every step of the synthesis process can be customized with the FlowTabs. To use the FlowTabs you merely walk through each tab in order while customizing along the way. This is essentially what the SynthesisWizard and Quick Setup accomplish for you with the default settings.

Conventions Used

The reader is alerted to terms, GUI names, and items with the following conventions:

- **Level 3** is shown in bold to emphasize that a particular instruction or GUI item is for **Level 3** only. For example, the main window for **Level 3** differs in appearance from Level 1 and Level 2 with the number of FlowTabs, the interactive command line shell, and in the banner area.
- **Level 3**: GUI command line in the Information Window is referred to as the **interactive command line shell**.
- Batch Mode is entered on the **DOS** or **UNIX** command line**.**
- The `Courier Font` is used for `file names,commands` and `variables`.
- **Buttons** and **keys** are typed in bold.
- Arrows indicate a menu or pulldown choice: Click File -> Open.
- Screen Shots: Example defaults, filenames, and field values are for illustration purposes only and may not apply to your particular synthesis task.
- FlowTabs refer to the series of tabs - Quick, Technology,....Output, P&R.
- Tab refers to a single tab - Output, for example.
- Power tab refers to the series of tabs that supports the FlowTabs - EDIF, VHDL, Verilog Options, for example.
- LMB is used for left mouse button.
- RMB is used for right mouse button.
- Choice boxes - ■ selected, ❑ not selected
- Radio buttons - ●selected, ❍not selected

PC Hardware and Software Requirements

These are the requirements for all levels of LeonardoSpectrum.

Type of PC

An IBM compatible PC with a Pentium or Pentium-Pro CPU is recommended. A 486 PC is acceptable, but may run slowly.

Operating System

LeonardoSpectrum requires Windows NT/95/98.

Disk Space

LeonardoSpectrum requires approximately 70 MBytes of disk space for programs and data files. Plan for an additional 50 MBytes for your files.

System Memory (RAM)

Table 1-1, System Memory, shows the recommended memory for proper operation of Exemplar synthesis tools. The actual requirements may vary; this depends on your design and coding style.

Table 1-1. System Memory Requirements

Design Size			
Number of Gates	Look Up Tables	Flip-Flops	RAM, MBytes
up to 15,000	up to 1100	500	64
15,000 to 75,000	1100 to 5000	3000	128
75,000 and up	5000 and up	5000	256

Note: A system running with less than the recommended memory may slowdown due to memory swapping.

PackagedPower

PackagedPower integrates Exemplar Logic's LeonardoSpectrum with Mentor Graphic's Renoir graphical design and management environment, and with Model Technology's ModelSim HDL simulator. PackagedPower is intended for engineers who are designing medium to large FPGAs or are in the process of moving across the FPGA/ASIC boundary.

The PackagedPower Environment allows you to launch and run both simulation and synthesis from Renoir and then to analyze results in LeonardoSpectrum for each operation through cross probing, cross highlighting, back annotation, and dynamic design animation.

Graphical User Interface 2

This chapter presents the LeonardoSpectrum GUI and describes the ways you can run the GUI components to suit your needs. Everything is here that you may need for a basic or a complex synthesis design. Information available from pulldowns, popups, tabs, fields, and buttons is just a click away. This chapter is divided as follows:

- Startup
- Main Window Description

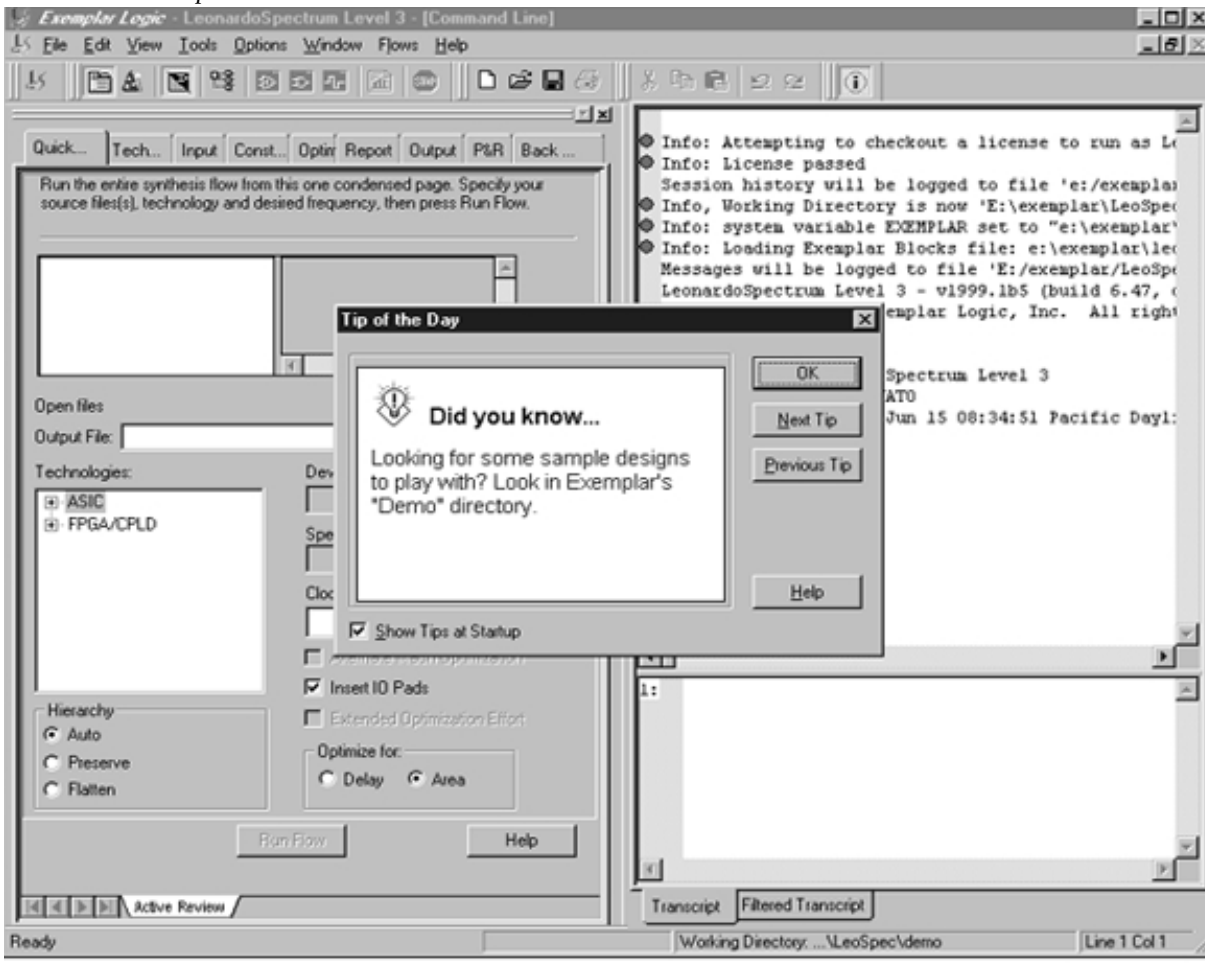
Startup

This section includes:

- Tip of the Day
- SynthesisWizard

When you start up LeonardoSpectrum for the first time, the main window is maximized and displays the Tip of the Day, FlowTabs, menu bar, toolbar, and an information window. Refer to Screen 2-1, Startup Main Window.

Screen 2-1. Startup Main Window



Tip of the Day

The tip of the day is a quick way to get important information. The tip of the day opens automatically on the first invocation of LeonardoSpectrum. Click forwards or backwards to move through the tips. If desired, change ■ Show Tips at Startup to prevent this window from opening. **Note:** The functions on the main window are not available until you close Tip of the Day.

SynthesisWizard

The SynthesisWizard consists of four steps that must be completed in the order presented. If you are a first-time user, then the SynthesisWizard is recommended to get you started right away. Continue to Chapter 13, SynthesisWizard Tutorial, for a description of each step.

The SynthesisWizard is one of three ways to synthesize your design; Quick Setup and FlowTabs are the other two ways.

Note: RMB over the FlowTabs to open this popup:

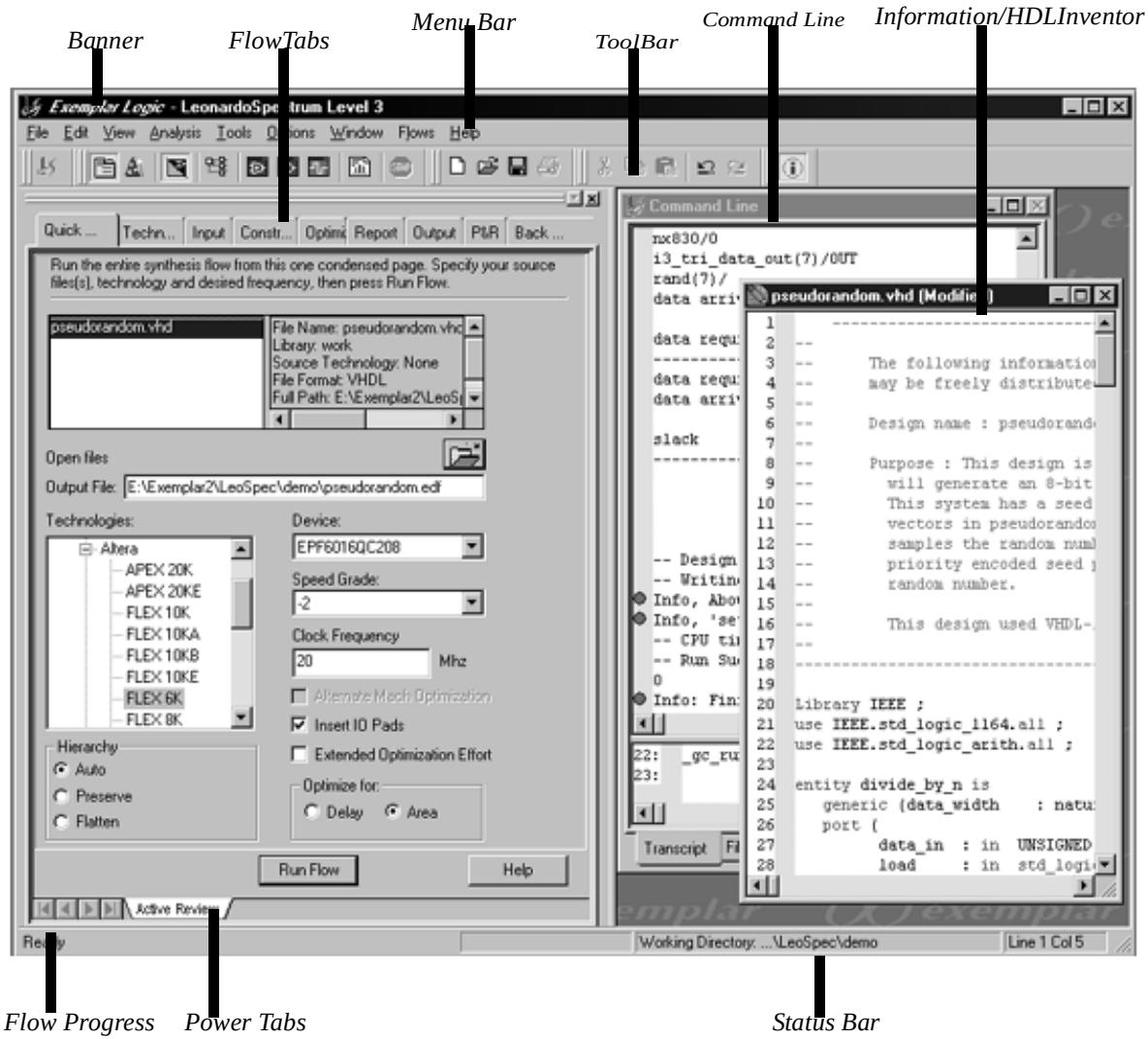
Allow Docking (rearrange windows and bars as needed)
Hide (turn off windows or bars as needed)
FlowTabs on left (tabs appear on left side of main window)
FlowTabs on top (tabs appear at top of main window, default)
Float in Main Window (float FlowTabs in main window)

Main Window Description

Refer to Screen 2-2. The main window is divided as follows:

- Main Window Header
- Information Window and HDLInventor
- Status Bar

Screen 2-2. Main Window

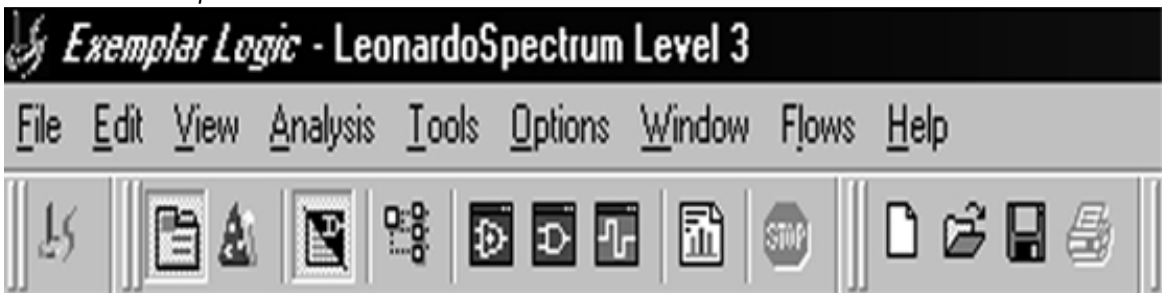


Main Window Header

Screen 2-3 shows a portion of the main window header. The items and icons on the header are described in the following tables:

- Table 2-1. Menu Bar Items
- Table 2-2. Toolbar Icons

Screen 2-3. Part of Main Window Header



Note: *Banner*: The banner identifies the Level of LeonardoSpectrum - Level 2 or Level 3

Table 2-1. Menu Bar Items

File Pulldown	The choices on the File pulldown allow you to manage and save files.
New (Ctrl+N)	File -> New opens an untitled window for a new file. This window is for entering your design code.
Open... (Ctrl+O)	File -> Open.... The Windows Open utility comes up. Files available under Files of type: are: • History File (*.his) • VHDL Files (*.vhdl;*.vhd;*.hdl) • Verilog Files (*.v;*.veri;*.h;*.ver) • TCL Files (*.tcl, *.scr) • Log Files (*.log) • EDIF Files (*.edif;*.edf; *.ed) • Report Files (*.sum) • XNF Files (*.xnf; *.x) • HDL Files (*.vhdl; *.vhd; *.hdl; *.v; *.veri; *.h; *.ver) • All Files (*.*)

continued...

Table 2-1. continued....

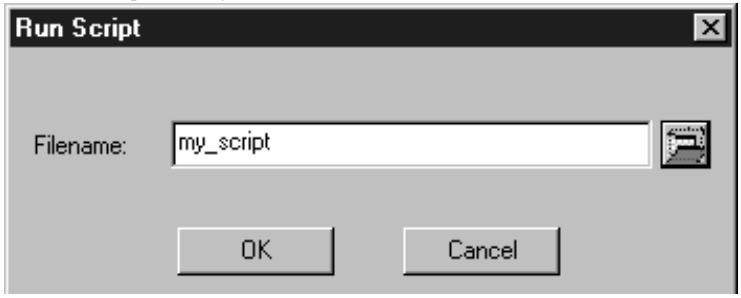
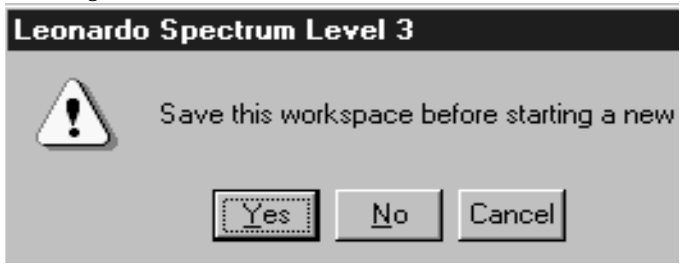
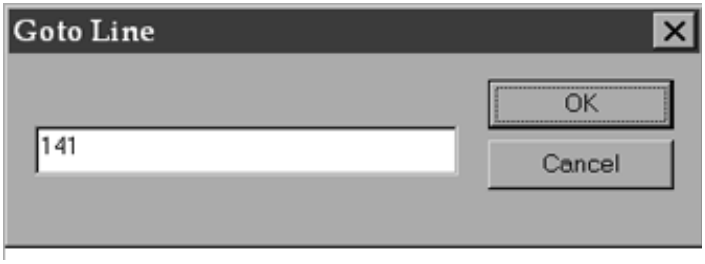
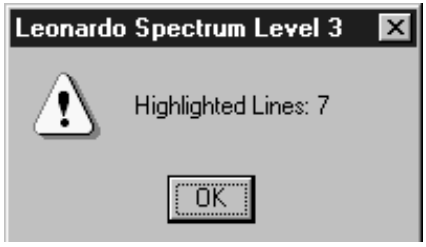
File Pulldown continued...	
Save (Ctrl+S)	File -> Save to save any file currently in the Information Window. If this is a new file, you are asked to specify a file name. Save overwrites the current active file with new information.
Save As... (Ctrl+A)	File -> Save As... to name a new file or rename a file. The Windows Save As utility opens. You can save as type: History File (*.his) or All Files (*.*)
Save Filtered Transcript	File -> Save Filtered Transcript to save your output file. This choice is available after you write the output file.
Run Script for Tcl **Note:** Refer to Chapter 6, Batch Mode Options, for information on sourcing your Tcl script from LeonardoSpectrum. The `group`, `unfold`, `analyze`, and `elaborate` commands are available **only** to **Level 3**.	File -> Run Script to open Run Script. Refer to the Command Reference Guide for Tcl script information. Click button on Run Script to open the Windows Open utility. Select a Tcl file (*.tcl) or All Files (*.*). 
New Project **Note:** If you click Yes, the Save Workspace As opens with unsaved_project.lsp in the File name: field. If you click No, then current settings are not saved.	File -> New Project to add a new project file *.lsp for LeonardoSpectrum. New project defaults to unsaved_project.lsp in the File name: field. You are not prompted for a project name and location. Before starting a new project, a check is made for any unsaved current project. If an unsaved project is found, you are prompted with “Save this workspace before starting a new one?” 

Table 2-1. continued....

File Pulldown continued...	The choices on the File pulldown allow you to manage and save files.
Open Project	File -> Open Project to bring up the Open utility. Files of type: LeonardoSpectrum Workspaces (Project) (*.lsp) is the default.
Save Project	File -> Save Project to bring up Save Workspace As for Files of type: LeonardoSpectrum Workspaces (Project) (*.lsp). You can also write over your current design with new information.
Save Project As	File -> Save Project As to bring up Save Workspace As for Files of type: LeonardoSpectrum Workspaces (*.lsp). When Save As opens, the current project name is already selected. You are prompted to confirm project or project file name and location. When you click **OK** the entire design is saved to your project folder or the default unsaved_project.lsp project folder. **Note:** v1999.x projects cannot be read by v1998.x unless you set the following variable in v1999.x to v1998.x: `xdb_write_version v1998.x`
Change Working Directory	File -> Change Working Directory. Use the standard directory navigator to set up your new Working Directory. The new Working Directory is saved as part of your design when you do Save or Save As. This working directory is the starting point for all relative pathnames, and will become the default output directory. Specify an absolute (not relative) pathname for your new working directory. Your current working directory is still displayed on the right side of the status bar.
Recent Files> (not available until after first invocation)	Path(s) to your recent file(s) after first startup.
Recent Projects> (not available until after first invocation)	Path(s) to your recent project(s) after first startup.
Exit	File -> Exit to exit LeonardoSpectrum. You are prompted to confirm.
Edit Pulldown	The Edit pulldown provides you with a list of Windows editing commands. The availability of these items depends on the activity on the main window. For example, an active HDLInventor enables the editing commands:
Undo (Ctrl+Z)	Edit -> Undo to reverse the last action.
Cut (Ctrl+X)	Edit -> Cut to remove selected text and place on clipboard.
Copy (Ctrl+C)	Edit -> Copy to copy text from clipboard to cursor position.
Paste (Ctrl+V)	Edit -> Paste to paste text from clipboard at cursor position.

continued...

Table 2-1. continued....

Edit Pulldown continued...	
Clear (del)	Edit -> Clear to delete selected text.
Select All (Ctrl+A)	Edit -> Select All to select all text.
Find (Ctrl+F)	Edit -> Find to find typed text. Search through files for specific data.
Find Next F3	Edit -> Find Next to find the next occurrence of a word or phrase.
Replace (Ctrl+H)	Edit -> Replace to replace text.
Goto Line	Edit -> Goto Line to open **Go to line**. Enter line number from displayed source code in HDLInventor. 
Get Highlighted Lines	Edit -> Get Highlighted Lines to open message with Highlighted line number. Click OK to bring line number into view in the displayed HDL source code. 

continued...

Table 2-1. continued....

View Pulldown	The View pulldown gives you choices to enable or disable the displays of: • Toolbar • Status Bar • Report Window
Analysis Pulldown	The Analysis pulldown is available when the HDLInventor is active. Analysis allows you to: • Trace to Hierarchy (Cross probe from source code to schematic) • Show next (currently unavailable)
Tools Pulldown	The Tools pulldown provides you with the following options:
Design Browser	Tools -> Design Browser to display ports, nets, instances, registers, and primitive cells. Refer to Chapter 8, LeonardoInsight.
View RTL Schematic (original, unmapped design)	Tools -> View RTL Schematic - LeonardoInsight provides you with a view of your original RTL schematic in the schematic viewer. Refer to Chapter 8, LeonardoInsight. **Note:** Before you can bring up the schematic viewer you must have an active design.
View Gate-Level Schematic	Tools -> View Gate-Level Schematic - LeonardoInsight provides you with a view of your gate level design in the schematic viewer. Refer to Chapter 8, LeonardoInsight. **Note:** Before you can bring up the schematic viewer you must have an active design.
Variable Editor	Tools -> Variable Editor. Refer to Chapter 11, Menu Bar Items.
Renoir Crossprobe	Tools -> Renoir Crossprobe. Productivity gains can be achieved by using the Renoir new cross referencing mechanism and hierarchical Find & Replace feature.
Options Pulldown	Options pulldown provides you with tabs for session settings and the browser filter. Refer to Chapter 11, Menu Bar Items, for information. • Session Settings... • Browser Filter...
Window Pulldown	Window pulldown provides you with task bar properties from Windows to organize and manage several open windows: • Arrange All • Cascade • Tile Horizontal • Tile Vertical • Information - Read Only (Retrieve Information Window.)

continued...

Table 2-1. continued....

Flows Pulldown	Flows pulldown provides you with the following: • SynthesisWizard (toggle steps 1 to 4) - Refer to Chapter 13. • FlowTabs (toggle between FlowTabs and Command Line)
Help Pulldown	Help pulldown provides you with:
Help -> Help Contents	Opens a series of tabs that are designed around the Windows properties. Help has indexes and context-sensitive choices. Help can guide you through the entire synthesis process.
Help -> Show Extended Help	This is help text at the top of the SynthesisWizard and FlowTabs.
Help -> Purchase	currently unavailable
Help -> Tip of the Day	Enable or disable Tip of the Day.
Help -> Video Tutorial	Open video tutorial.
Help -> View User Manuals	List of available pdf documents.
Help -> About	Opens a display of the LeonardoSpectrum version number, Level information, and Copyright. Use this information when contacting technical support.

Table 2-2. Main Window Header Tool Bar Icons

Exemplar's Logo	Click -> logo to open your system browser (for example Netscape). The web address is: Exemplar Logic, http://www.exemplar.com
Task Icons	The icons can be moved in groups to suit your needs. LMB over an icon to popup a short balloon title. The entire icon title appears in the status bar.
Show Hide Flow Bar	Toggle
A step by step wizard for the synthesis of your design	Refer to SynthesisWizard in Chapter 13.
Enable Cross Probe	Refer to Chapter 8.
Design Browser	Refer to Chapter 8.
View RTL Schematic	Refer to Chapter 8.
View Technology Schematic	Refer to Chapter 8.
View Critical Path Schematic	Refer to Chapter 8.
View the current summary file	Opens the review window.
Interrupt the current run	STOP - red, when completed STOP is grayed out.
Editing Icons	The editing icons are available when the HDLInventor is active.
Create a new document	Same function as File -> New.
Open an existing document	Same function as File -> Open.
Save the active document	Same function as File -> Save.
Print the active document	Print with Windows utilities.
Cut the selection and put it on the clipboard	Same function as Edit -> Cut.
Copy the selection and put it on the clipboard	Same function as Edit -> Copy.

continued...

Table 2-2. continued...

Editing Icons, continued...	
Paste/insert clipboard contents	Same function as Edit -> Paste.
Undo the last action	Same function as Edit -> Undo.
Redo the previous undone action	Same function as Edit -> Redo.
Show extended help	Show extended help on the SynthesisWizard and FlowTabs pages.

Information Window

This area shows the full report for your synthesis run and the coding of your design.

- Command Line Window
- HDLInventor

Command Line Window

The Command Line Window opens at startup. Refer to Screen 2-4. This window is for read-only messages, reports, and interactive command line shell. If closed, the window opens again when new information arrives. The interactive command line shell, `Leonardo 1:`, is available only for **Level 3**.

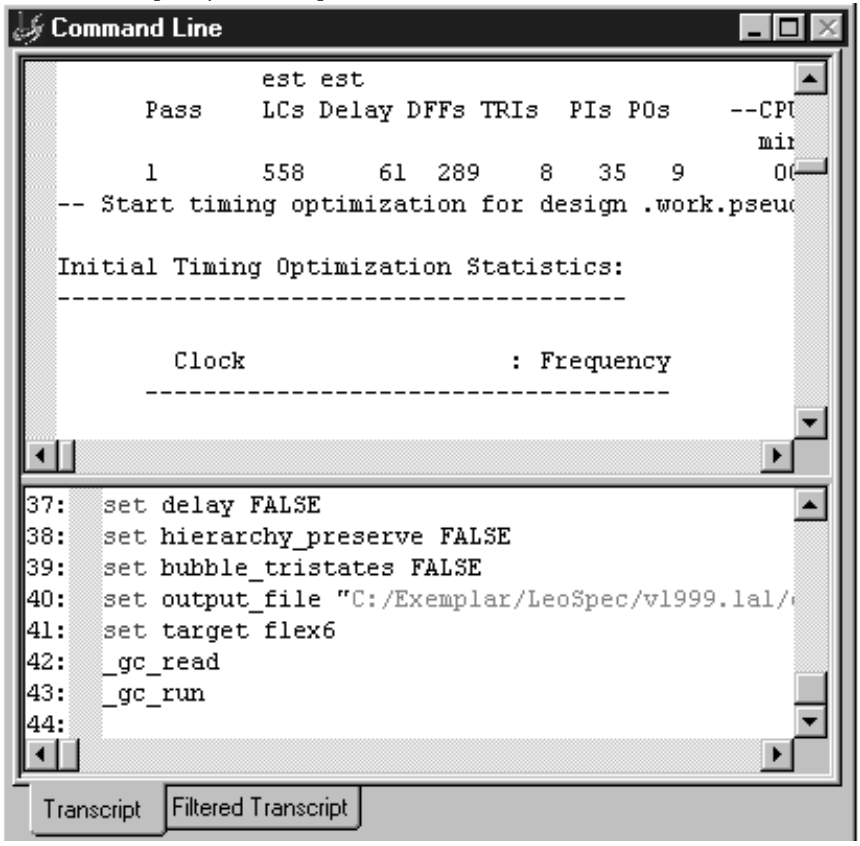
Note: The following is a portion of the Sample Script from Chapter 12, Design Methodology:

```
# Loading Target Technology
load_library cg61

# Setting operating conditions
set temp 80
set process typical
set voltage 5.0

# Setting Design Rule Conditions
set max_fanout_load 16
set max_cap_load 4
set max_transition 1.2
set wire_tree worst
```

Click Filtered Transcript to expand the Information - Read Only window. Green, red, and blue buttons may appear in the left gutter (margin). You can double click a blue button to bring up the HDLInventor.

- Red - Error
- Green - Information
- Blue - Warning

Screen 2-4. Example of a Full Report



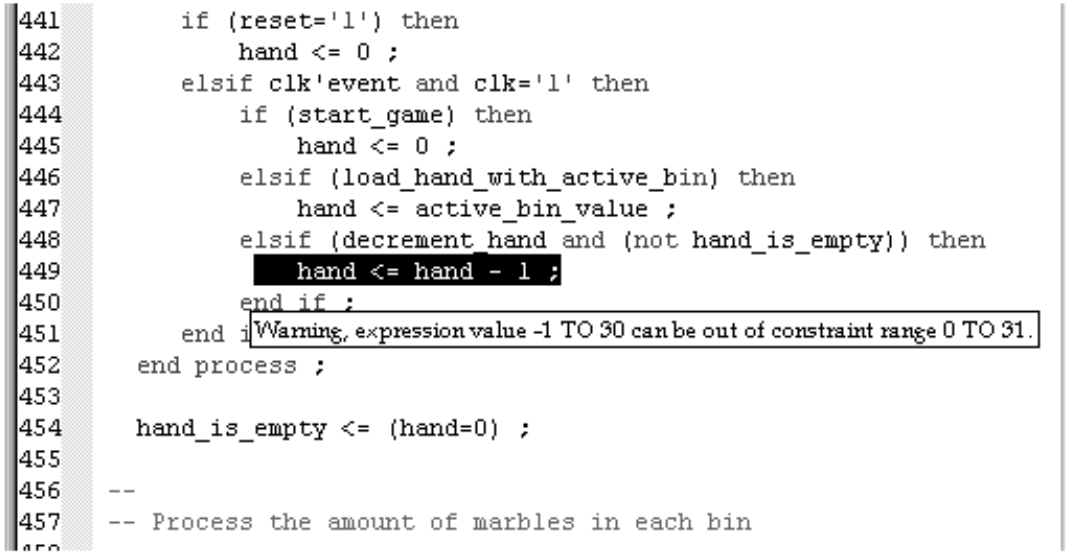
HDLInventor

Note: Double click LMB over input file name on Quick Setup or Input FlowTabs to open HDLInventor in the information window.

The HDLInventor is an interactive source code editor. The errors of syntax constructs found during synthesis are highlighted in distinctive colors. You can easily interpret the color (red, green, blue) for the type of warning or error. Errors, information, and warnings are annotated directly to the integrated HDL source code editor, HDLInventor. The source code editor is linked to the transcript in the message and report window. Line numbering identifies the line number in the source code. An information message pops up as you move the mouse cursor over the line number. Refer to Screen 2-5.

Note: Actual line numbers may differ from examples in Screen 2-5.

Screen 2-5. Part of Source Code with Popup Message


441 if (reset='1') then
442 hand <= 0 ;
443 elsif clk'event and clk='1' then
444 if (start_game) then
445 hand <= 0 ;
446 elsif (load_hand_with_active_bin) then
447 hand <= active_bin_value ;
448 elsif (decrement_hand and (not hand_is_empty)) then
449 hand <= hand - 1 ;
450 end if ;
451 end i
Warning, expression value -1 TO 30 can be out of constraint range 0 TO 31.
452 end process ;
453
454 hand_is_empty <= (hand=0) ;
455
456 --
457 -- Process the amount of marbles in each bin

Templates

The HDLInventor includes a list of templates that are predefined RTL templates. You can instantiate a template directly into your HDL source code. You can also create custom templates.

Note: LMB over highlighted file in the input window and double click to open HDLInventor. This editor also contains a set of VHDL and Verilog templates that include a macro template library of state machines, counters, ALUs, and technology specific comments. The editor allows you to trace syntax errors directly back to your source code for quick and easy debugging. For convenience, Table 2-3 lists the builtin templates.

Table 2-3. Builtin Templates

Template	Function
Overall Structure	context clauses library units
Architecture Body	_architecture_name _entity_name _signal_name
Finite State Machine	State Machine with Asynch Reset State Machine without Asynch Reset
Full Designs	Full Design Counter Full Design flip flop Full Design 3-State Buffer
Statements	Case Statement Component Declaration Component Instantiation Statement Concurrent Procedure Call Concurrent Signal Assignment Statement Conditional Signal Assignment Constant Declaration Entity Declaration For Statement Generate Statement (for generate) Generate Statement (if generate) If Statement Library Clause Package Declaration Procedure Call Statement Process (combinatorial logic) Process (sequential logic) Selected Signal Assignment Statement Signal Declaration Signal Assignment Statement Subtype Type USE Clause Wait Statement Variable Declaration Statement Variable Assignment Statement

Editing Options - HDLInventor

You can add templates, do edits, and toggle to add or remove bookmarks. Use these steps:

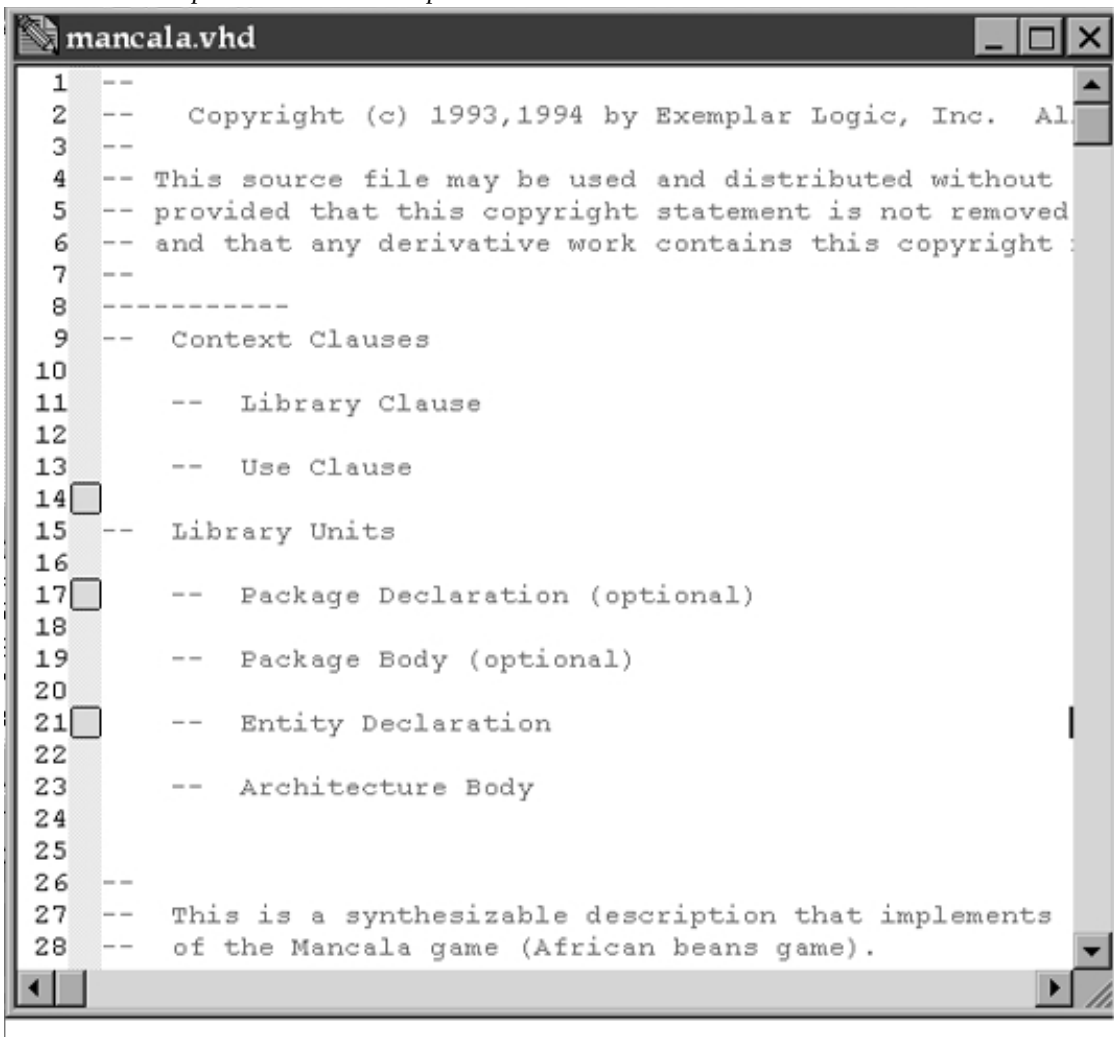
1. RMB over the HDLInventor to open this popup:
   - Undo, Redo, Cut, Copy, Paste
   - Toggle bookmark
   - Open Report Window
   - View line numbers
   - Insert template> (Refer again to Templates in this section.)
2. Highlight the desired code to apply Windows edit functions.
3. Click line of code for placing bookmark next to line number. Click again to remove bookmark. Refer to Screen 2-6.
4. Click View line numbers to toggle line numbers on and off.
5. Click Insert template> to bring up the template list. Click to select and insert list. The template requires editing. Refer again to Table 2-3.

Editing Options - Transcript and Filtered Transcript

You can toggle to add or remove bookmarks and to turn messages on and off, for example. Use these steps:

1. RMB over left margin of either the Transcript or Filtered Transcript to open this popup:
   - Toggle Bookmark
   - View Line Numbers
   - Save (Transcript only)
   - Save Filtered Transcript
2. RMB over left margin of the Filtered Transcript to open this popup list:
   - Show Transcript
   - Show Errors
   - Show Warnings
   - Show Information
   - Show Commands

Screen 2-6. Example HDLInventor Template and Bookmarks

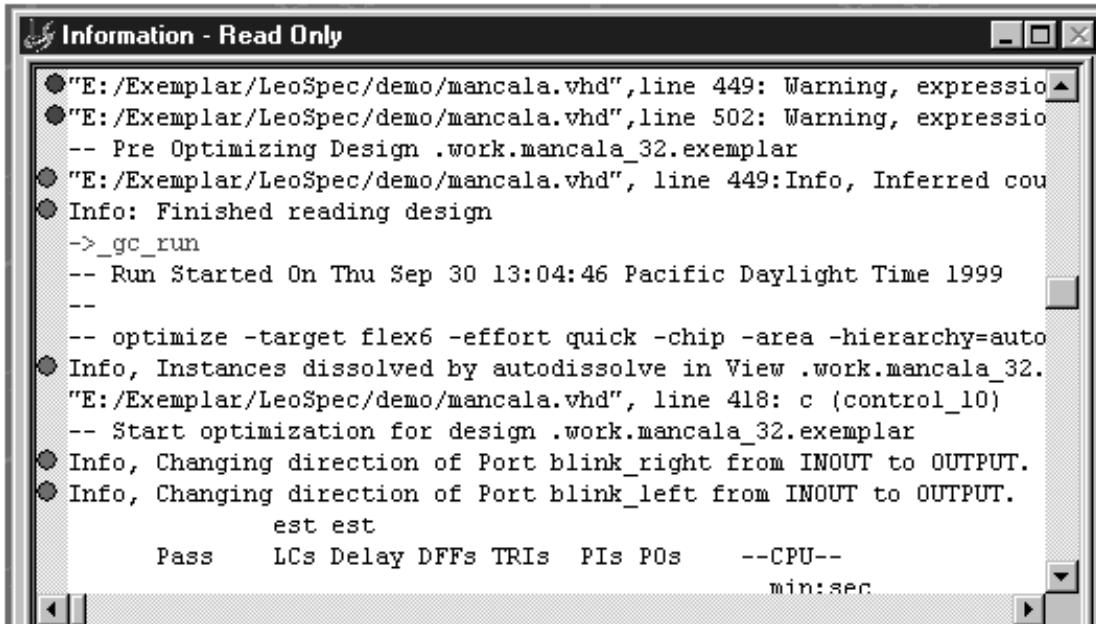


More Editing Options

Use these steps to edit your code:

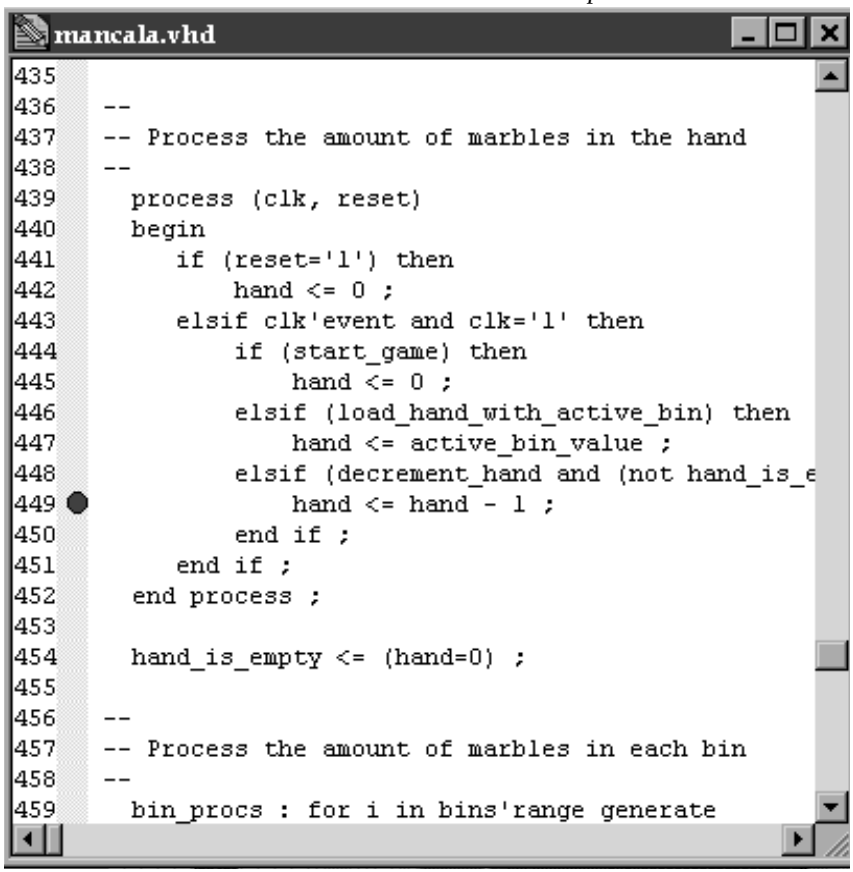
1. Click or double click on a red, green, or blue button. Refer to Screen 2-7.
2. Click to bring up line numbers. Edit source code as needed.

Screen 2-7. HDL Source Code Example


Information - Read Only
"E:/Exemplar/LeoSpec/demo/mancala.vhd",line 449: Warning, expressio
"E:/Exemplar/LeoSpec/demo/mancala.vhd",line 502: Warning, expressio
-- Pre Optimizing Design .work.mancala_32.exemplar
"E:/Exemplar/LeoSpec/demo/mancala.vhd", line 449:Info, Inferred cou
Info: Finished reading design
->_gc_run
-- Run Started On Thu Sep 30 13:04:46 Pacific Daylight Time 1999
--
-- optimize -target flex6 -effort quick -chip -area -hierarchy=auto
Info, Instances dissolved by autodissolve in View .work.mancala_32.
"E:/Exemplar/LeoSpec/demo/mancala.vhd", line 418: c (control_10)
-- Start optimization for design .work.mancala_32.exemplar
Info, Changing direction of Port blink_right from INOUT to OUTPUT.
Info, Changing direction of Port blink_left from INOUT to OUTPUT.
est est
Pass LCs Delay DFFs TRIs PIs POs --CPU--
min:sec

Note: Example lines 449 and 502 in Screen 2-7 (“line 449: Warning,...”; line 502: Warning,...”) are highlighted with a blue button. Double click on button 449 with the left mouse button to open the HDLInventor. Screen 2-8 shows the lines of code for line number 449. Red button indicates a warning and the green button indicates information.

Screen 2-8. HDL Code with Editor Line Number Example


mancala.vhd
435
436 --
437 -- Process the amount of marbles in the hand
438 --
439 process (clk, reset)
440 begin
441 if (reset='1') then
442 hand <= 0 ;
443 elsif clk'event and clk='1' then
444 if (start_game) then
445 hand <= 0 ;
446 elsif (load_hand_with_active_bin) then
447 hand <= active_bin_value ;
448 elsif (decrement_hand and (not hand_is_e
449 hand <= hand - 1 ;
450 end if ;
451 end if ;
452 end process ;
453
454 hand_is_empty <= (hand=0) ;
455
456 --
457 -- Process the amount of marbles in each bin
458 --
459 bin_procs : for i in bins'range generate

Status Bar

The status bar provides you with the following information:

- Ready message, Toolbar messages, Flow Progress messages
- Current Working Directory
- Line Counter

Level 2 FlowTabs 3

FlowTabs are available after every startup. These tabs plus the power tabs bring more options to your design. The FlowTabs are designed for the advanced user who needs access to all the embedded power of LeonardoSpectrum. Nearly every step of the synthesis process can be customized based on FlowTabs. To use the FlowTabs you merely walk through each tab in order while customizing along the way. This is essentially what the synthesis wizard and Quick Setup accomplish for you with default settings.

Note: Click **OK** to close Tip of Day, if necessary. Click **Help** on FlowTabs for assistance.

Note: Refer to Additional Instructions in Chapter 4.

Synthesis FlowTabs

The FlowTabs and power tabs guide you through the synthesis process. This chapter is divided as follows:

- Quick Setup Tab
- Technology Tab
- Input Tab
- Constraint Tab
- Optimize Tab
- Output Tab
- P&R Tab: Altera FLEX/Quartus, Xilinx, Lattice/Vantis
- Back Annotation Tab

Quick Setup Tab Active Review - FPGA

Quick Setup is intended for the user who is familiar with LeonardoSpectrum and the synthesis process. Everything which can be specified in the synthesis wizard, can be specified on one condensed tab. Once specified, you can hit the run button to run the entire synthesis flow including synthesis, global constraints, optimization and writing netlist. In addition, Quick Setup automatically sets up all options, defaults, and settings in the FlowTabs to assist you when walking through the more advanced tabs.

Note: Refer also to Chapter 4, Retarget Steps for an Output Netlist.

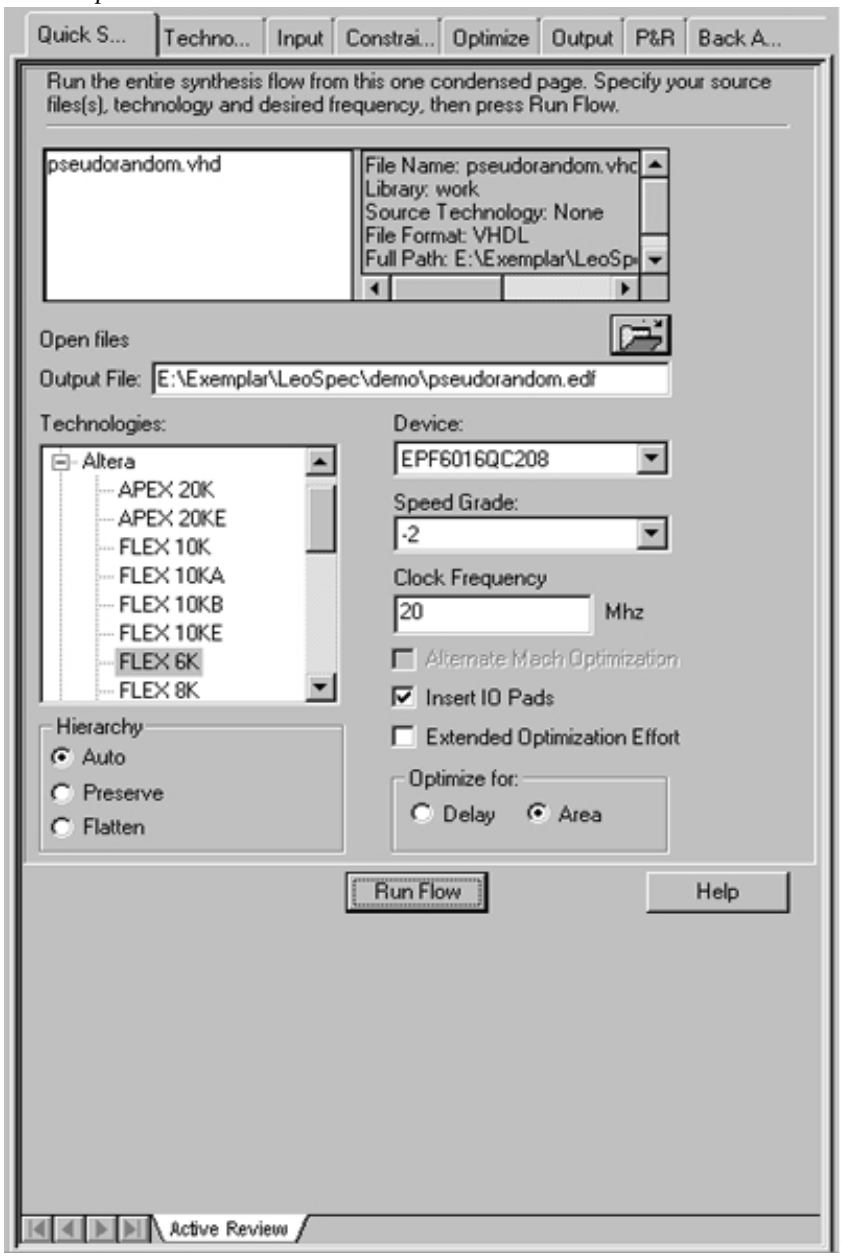
Note: Refer to Chapter 5, Reports, for applying a constraint file to Quick Setup.

Refer to Screen 3-1, Quick Setup, Active Review. Active Review is a condensation of the SynthesisWizard. Refer to Chapter 13, SynthesisWizard Tutorial. In addition, the options shown in Table 3-1 are also available:

Table 3-1. Part of Quick Setup Options

Option	Description
❑ Alternate Mach Optimization	Lattice/Vantis only: Click to apply alternate synthesis heuristics. This flow may produce better results on small designs.
Optimize for:	❍Delay - The design is faster and the area may be bigger. ●Area - The design is slower and the area may be smaller.
■Insert I/O Pads	This box is selected by default. LeonardoSpectrum runs the optimization in the chip mode and inserts I/O pads in your design. If this box is not selected, then LeonardoSpectrum runs the optimization in the macro mode.
❑ Extended Optimization Effort:	If this box is selected, then LeonardoSpectrum runs additional optimization algorithms.
●Hierarchy Auto (auto-dissolve)	●Hierarchy Auto is selected by default. Views containing 3000 instances or less are dissolved. If ❍Hierarchy Auto is not selected, then instances are not dissolved. Refer also to Chapter 4, Additional Instructions.
❍Hierarchy Preserve	If ❍Hierarchy Preserve is not selected then your design is flatten before optimizing. If ●Hierarchy Preserve is selected then your design hierarchy is not changed during optimization. Refer to the Command Reference Guide for boundary optimization information.
❍Hierarchy Flatten	If selected then your entire design hierarchy is flattened.
Click **Run Flow** to start flow.	

Screen 3-1. Quick Setup - Active Review



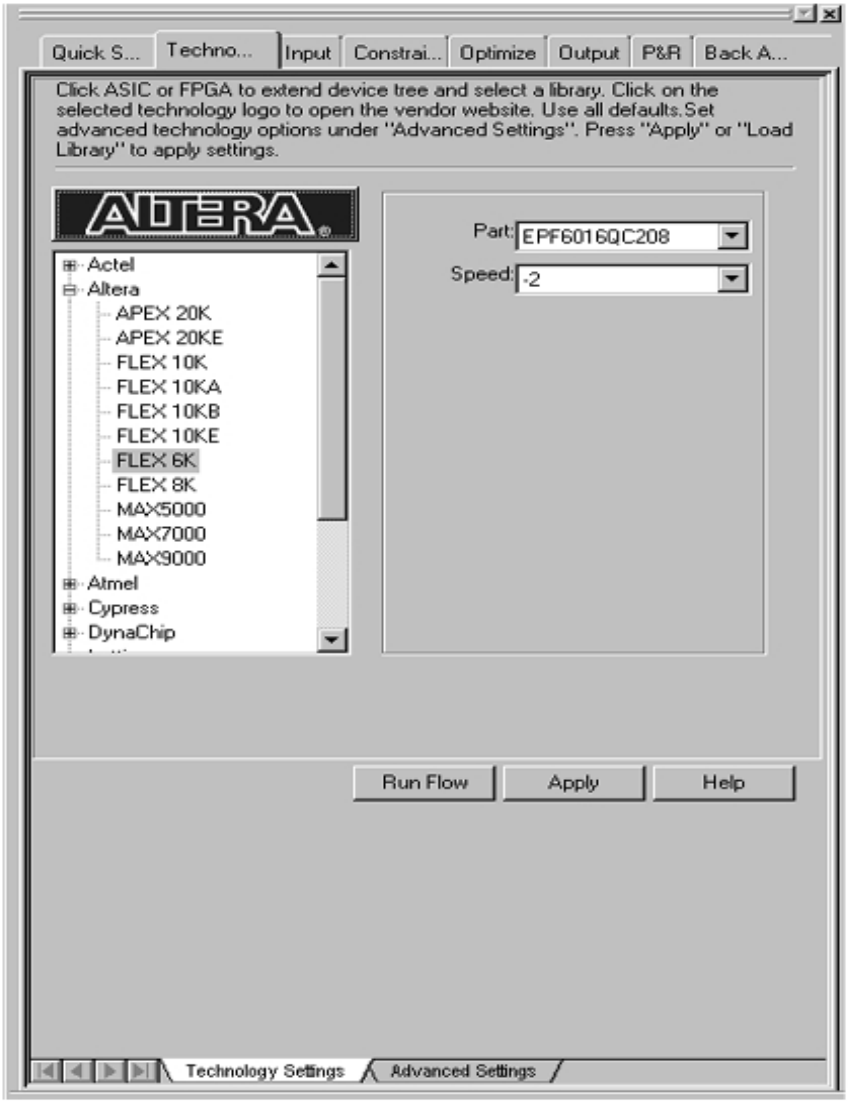
Technology Tab - FPGA

Refer to Screen 3-2, FPGA Technology Settings. Refer to Table 3-2.

Table 3-2. Technology Settings

Option	Description
Part:	This is the part number of your target device.
Speed:	This the speed grade (process) of your target device.
Click on **Apply** to apply your options and **Run Flow** to run the flow	

Screen 3-2. Technology Settings



Input Tab

Refer to Screen 3-3, Input Files. This input tab is identical to the input steps for the SynthesisWizard in Chapter 13. Refer to Chapter 7, Power Tabs and Advanced Topics for the power tab screens for VHDL, Verilog, XNF, and EDIF. Refer to Chapter 6, Batch Mode, for Encoding Style (Binary, Onehot, Random, Gray, Twohot, Auto) information.

Note: Click on filename to fill in the information in the right window. Notice that the Source Technology: is None. When you retarget your output netlist to another technology, the Source Technology is the name of the technology in your output netlist. Refer to Chapter 4, Retarget Steps for Output Netlist.

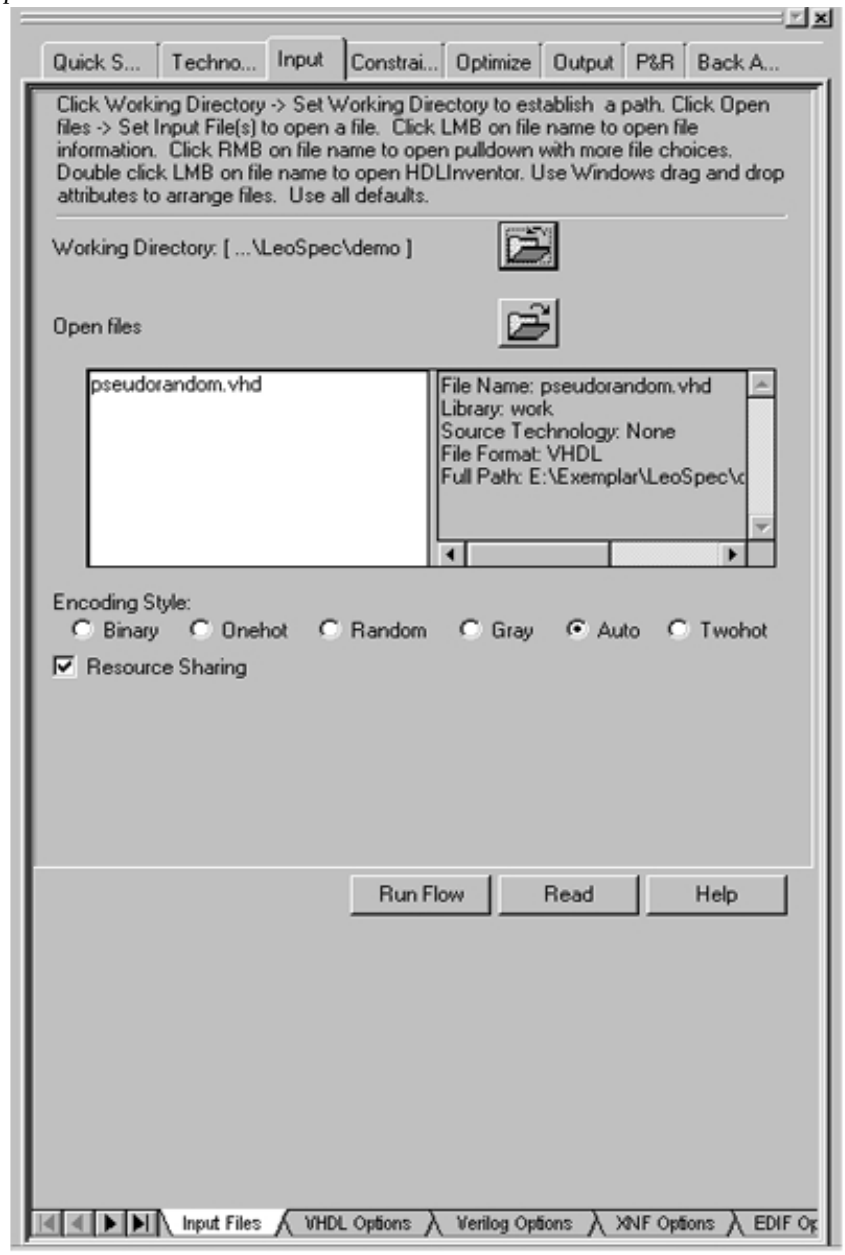
Note: Use Windows attributes to drag and rearrange files as required in the Open files window. Double click on an input file name to open the HDLInventor.

Note: RMB over your input file to pop up these shortcuts:

Add Input File: (Opens Set Input Files.)
Reverse Order
Toggle Selection
Select All
Open File: (Opens file in the Information Window.)
Set Work Library: (Opens Change work library, type in name.)
Set Technology: (Opens a list of technologies.) **Note Xilinx:** Simprim and Neoprim library components are available in this list.
Set File Type: (Opens lists of output formats.)
Remove: (Click to remove highlighted input file.)
Remove All: (Click to remove the entire workspace.)

Click **Read** to read your design into the database. Click **Run Flow** to start the synthesis flow.

Screen 3-3. Input Files



Constraints Tab

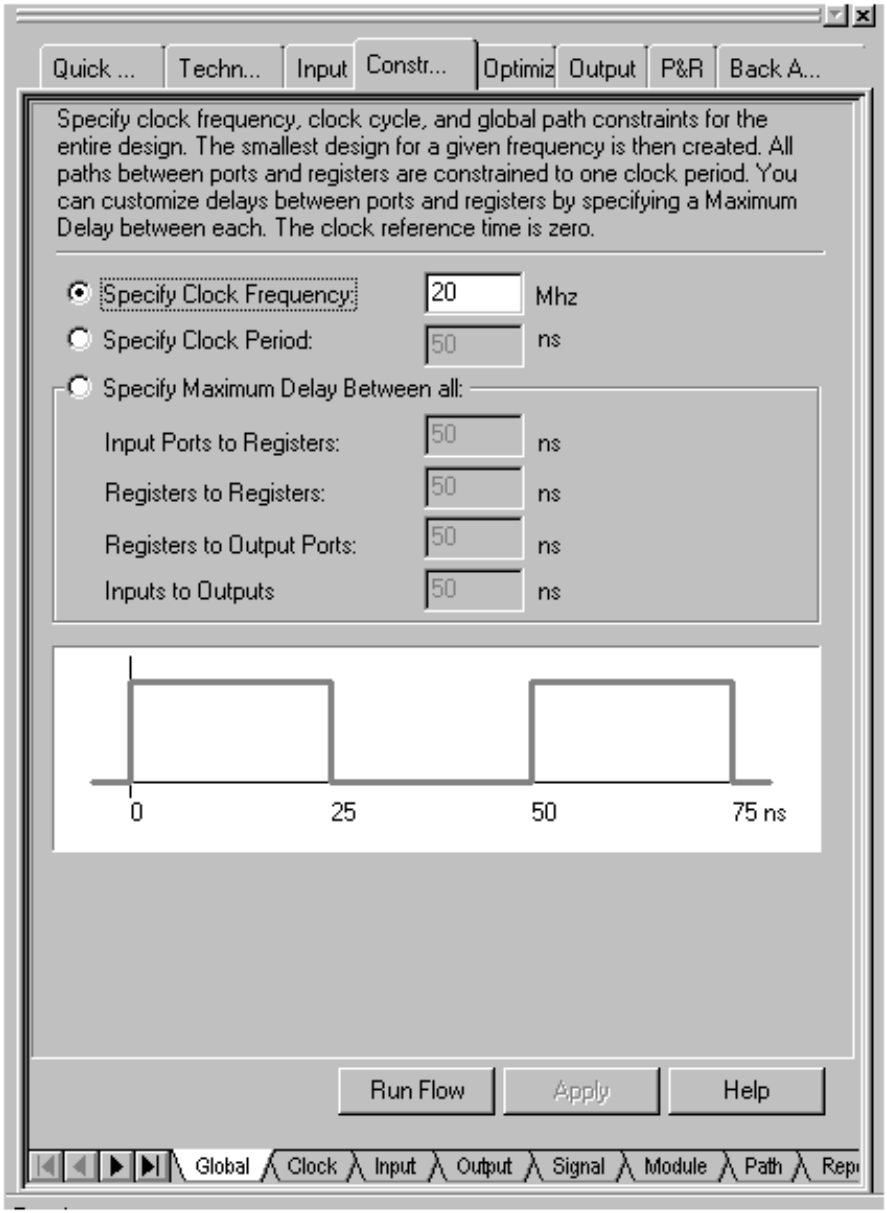
Refer to Screen 3-4, Global Constraints. The eight scrolling power tabs are explained in Chapter 5, Constraint Editor. Table 3-3 is duplicated here from Chapter 5 for convenience.

Click **Run Flow** to start the flow.

Table 3-3. Global Constraints

Option	Description
●Specify Clock Frequency, (Mhz):	This radio button choice is mutually exclusive with Clock Period. You can specify the required frequency for your design which is 1/period.
❍Specify Clock Period, (ns):	This radio button choice is mutually exclusive with Clock Frequency. See clock period in diagram.
❍Specify Maximum Delay Between all:	This radio button choice gives you control over the delays from port to register, register to register, register to port, and port to port.
Input Ports to Registers:	Delay from input port to input of register in nanoseconds.
Registers to Registers:	Delay from output of one register to input of another register in nanoseconds.
Registers to Output Ports:	Delay from output of register to output port in nanoseconds.
Inputs to Outputs	Delay from input port(s) to output port(s).
Waveform Window	The clock pulses show your settings.

Screen 3-4. Global Constraints

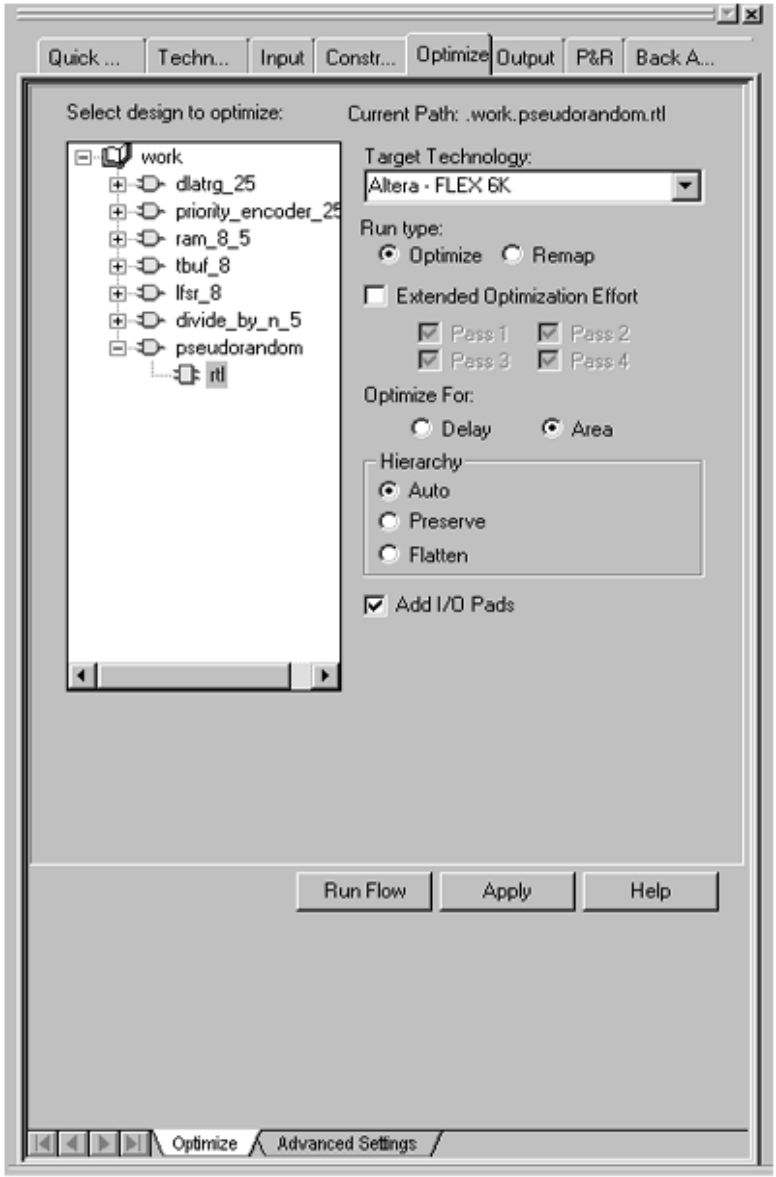


Optimize Tab

Table 3-4. Optimize

Option	Description
Select design to optimize: Click to select an object from the filtered, embedded design browser tree to be optimized. This tree was built when you read in your design(s). Refer also to Chapter 7, Advanced Settings.	
Current Path: The design you select from the design browser tree or your present design is the Current Path: Target Technology: Your current target is in the window.	
Run type: Control your run.	●Optimize: Default. Runs multiple optimization passes. Optimize means to reduce and improve logic in your design in terms of area and delay. ❍Remap: Does not optimize the network, but maps it into the target technology. The target may be another technology.
❑ Extended Optimization Effort: If this box is selected, then LeonardoSpectrum runs an additional three optimization algorithms (assumes all 4 Pass boxes are selected). While selecting Passes (1-4) may cause a slower run, an improvement in the use of design space may occur. However, this may take 4 to 6 times longer to run when compared with a single pass. The Run type: ●Optimize must be selected.	
A report is made for:	❑ Pass 1 ❑ Pass 2 ❑ Pass 3 ❑ Pass 4
Optimize for:	❍Delay - The design is faster and the area may be bigger. ●Area - By default the design is slower and the area may be smaller.
■Add I/O Pads	This box is selected by default. LeonardoSpectrum runs the optimization in the chip mode and inserts I/O pads in your design.
❑ Add I/O Pads	If this box is not selected, then LeonardoSpectrum runs the optimization in the macro mode.
●Hierarchy Auto (auto-dissolve)	●Hierarchy Auto is selected by default Views containing 3000 or fewer instances are dissolved. If ❍Hierarchy Auto is not selected, then instances are not dissolved.
❍Hierarchy Preserve	If ❍Hierarchy Preserve is not selected then your design is flatten before optimizing. If ●Hierarchy Preserve is selected then your design hierarchy is not changed during optimization. Refer to Reference Guide for boundary optimization information
❍Hierarchy Flatten	If selected then your entire design hierarchy is flattened (dissolved).
Click **Apply** to apply options. Click **Run Flow** to start the flow.	

Screen 3-5. Optimize



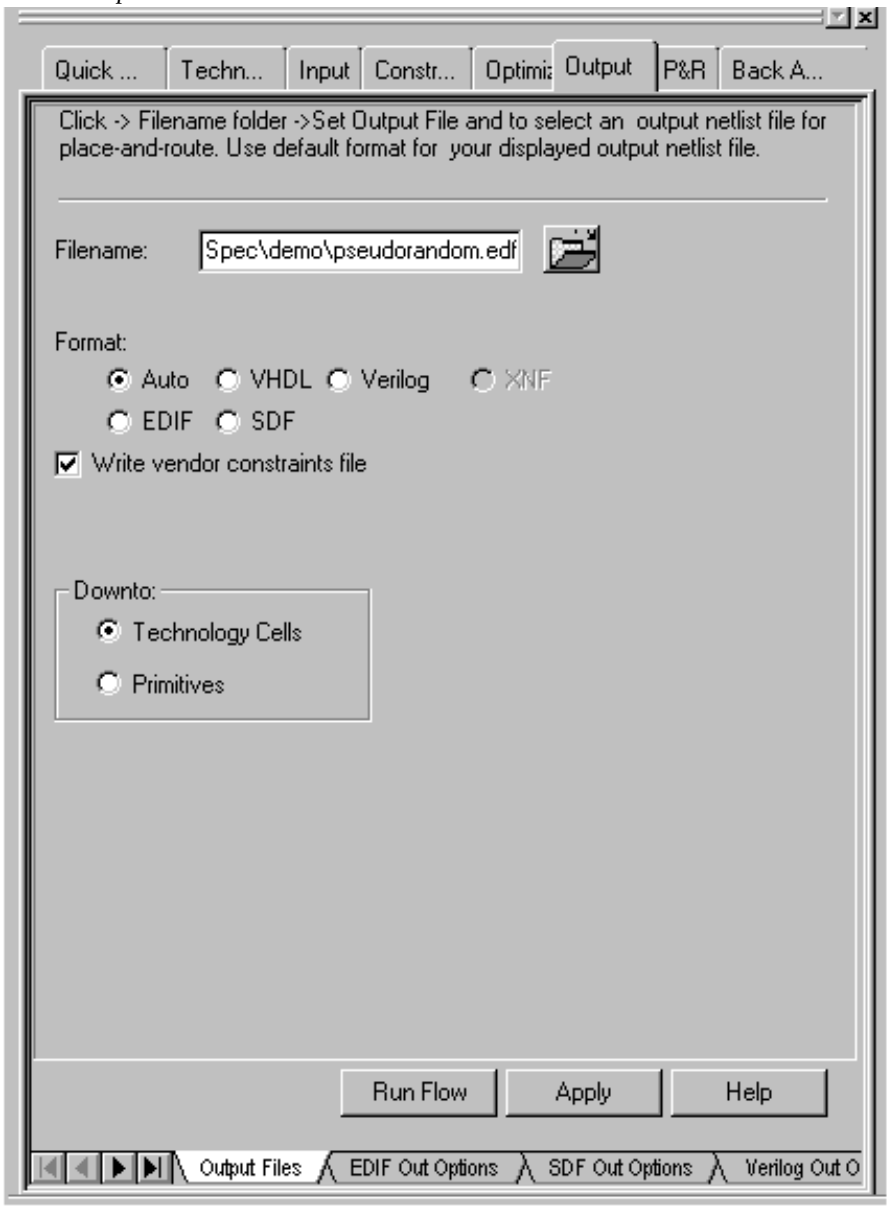
Output Tab

Refer to Screen 3-6 and Table 3-5 for output information. Refer to Chapter 7, for power tab options.

Table 3-5. Output Tab

Option	Description
Specify your output file parameters.	
🗁Filename: Note: Use a "-" for filename to have output appear on the main window.	Click on folder to open Set Output Files. This is the place-and-route file. Select from list or type in another filename. This filename defaults to <input design>.<ext>, where <ext> is based on the output format. **Note:** Point at filename to popup a bubble with full path name.
Format:	The radio button format choices are listed to meet your place-and-route technology needs.
●Auto	By default the output file is written out in EDIF format. Auto determines the actual format based on the filename extension. If the extension is .edf then Auto applies the EDIF format.
❍VHDL	The output design file is in the VHDL netlist format.
❍Verilog	The output design file is in the Verilog netlist format.
❍XNF	XNF is enabled only for Xilinx.
❍EDIF	The output design file is in the EDIF netlist format.
❍SDF (Standard Delay Format)	The output file is in the SDF netlist format. Output files in SDF are accepted by all technologies. This is a back-annotated SDF file.
■Write vendor constraints file:	Selected by default to write output file and a vendor's constraint file.
Downto:	●Technology Cells: Output file includes technology cells. ❍Primitives: Output file includes your original design.
Click: **Apply** to apply your options; **Run Flow** to run the flow.	

Screen 3-6. Output Files



P&R Tab

Table 3-6. Place and Route - Altera Description

Option	Description
Refer to Screen 3-7. The P&R tab information is available after you load the Altera library, complete the design flow, and write an output netlist file. **Note:** The run PR button is available only when your target technology is Altera.	
■Setup MAX+PLUS II Create (Assignment and Configuration File) ACF file: This option is selected by default. ❑Setup MAX+PLUS II allows you to change or overwrite an existing ACF file.	
❑Auto Fast I/O	Select to allow MAX+PLUS II compiler to implement registers in Fast I/O. May reduce area requirements but can slow internal circuitry.
❑Auto Implement in EAB (Altera FLEX 10K)	Select this box if you are using wide gates and you want to embed the array block (EAB).
❑Auto Register Packing	If your registers always have a constant input, for example “1”, then these registers are merged in the EDIF. Implements register packing by placing a combinational logic function and a register with a single data input in the same logic cell.
■Run MAX+PLUS II	Select this box if you want to run MAX+PLUS II for your EDIF file.
❑Bring up MAX+PLUS II	Select this box if you want to bring up MAX+PLUS II GUI. Otherwise run in batch mode.
■Timing Analysis	Use the timing information in the SDF, VHDL, or Verilog file to check place and route for accuracy. Creates either an input to output delay matrix, a setup/hold matrix, or a register performance report.
❍Input-Output Delay	Select for a typical delay.
❍Setup/Hold	Select if you want to check on setup and hold violations.
●Register Performance	Select if you want to verify that constraints are met.
Path to MAX+PLUS II executable	This is the path to the Altera script 🗁to invoke MAX+PLUS II.
Write Output For:	■Back Annotated Timing Analysis
❑Simulation: When selected, LeonardoSpectrum instructs Alliance Series to generate VHDL or Verilog simulation netlists targeting the simprims primitive cell set, and to generate an SDF file for backannotated timing.	❍VHDL ❍Verilog
Click **Run PR** to create your ACF file and to invoke MAX+PLUS II using specified options.	

Screen 3-7. Place and Route - Altera.

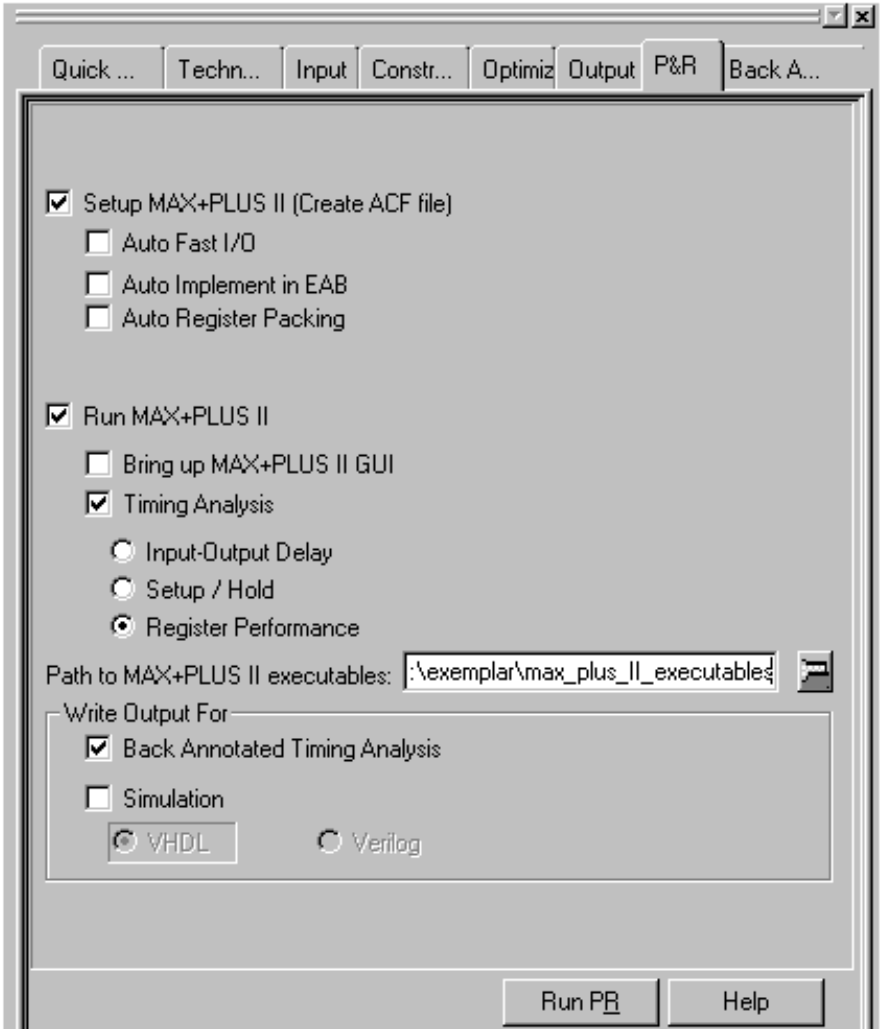


Table 3-7. Place and Route - Xilinx

Option	Description
Setup your place and route options and invoke the Xilinx M1 tools. This P&R tab is available after you load the Xilinx library, complete the design flow, and write an output netlist file.	
● Execute Place Route - Startup the DesignManager	
Effort	●Standard Uses standard effort to perform place and route. ❍High Performs place and run for better results than Standard, but takes a longer time to run.
❑Generate Files for timing simulation	After the EDIF place and route, then run the SDF, VHDL or Verilog files through back annotation to simulate the router design.
❑Generate bit file	Select this box to generate a bit file after place and route to program a Xilinx device. Not selected by default.
❑Use bitgen command file:	Select this box to open Bit Gen Command File🗁 or type in a file name. This file contains options for Xilinx bit file generator. Not a default.
❍Only generate a netlist for functional simulation.	Only generates a VHDL or Verilog file. A SDF timing file is not needed.
❍Only generate pre place & route delay estimate	Select if you want to generate and estimate of place and route instead of actually doing a place and route.
❍Run Design Manager	Select if you want to run the Xilinx M1 GUI.
❑Gate Level back annotation	Select this box if you want to do gate level with your EDIF file. Use an NGM file with ngdanno to perform gate level back annotation. Not a default.
Install path for Place_route exec:	This is the path to the directory 🗁where your Xilinx M1 executable is installed. If Xilinx environment variable is set or if the executable is in your path then leave this field blank.
Write Output For:	■Back Annotated Timing Analysis
❑Simulation: Alliance generates VHDL or Verilog simulation netlists to target the simprim or neoprim primitive cell set, and to generate an SDF file for back annotated timing.	❍VHDL ❍Verilog
Click **Run PR** to invoke Xilinx M1 tools using specified options.	

Screen 3-8. Place and Route - Xilinx

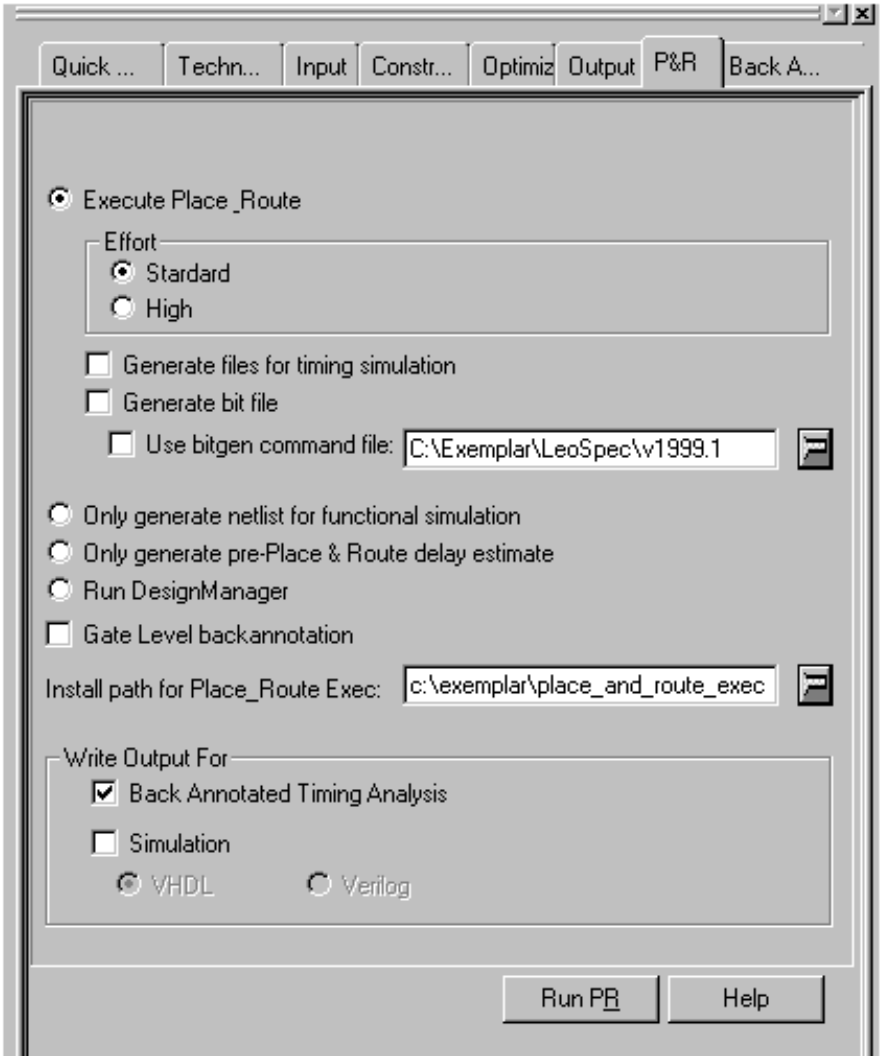
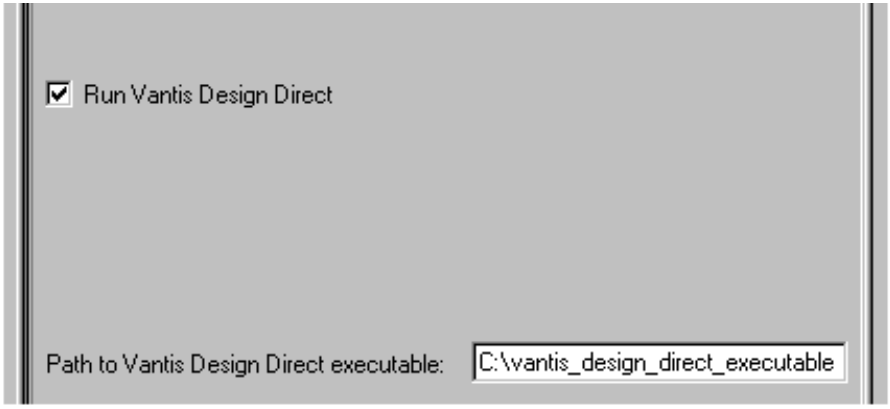


*Table 3-8. Place and Route - Lattice/Vantis**

Option	Description
■ Run Vantis Design Direct	Run the Vantis Design Direct GUI.

Type a path for your Design Direct executable.

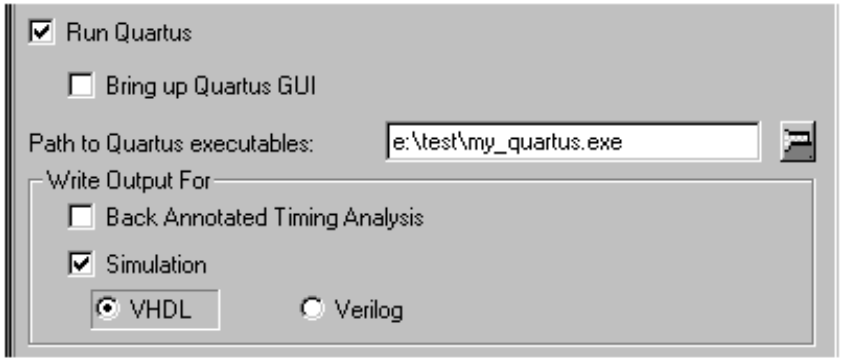
*Screen 3-9. Place and Route - Lattice/Vantis**



*Refer also to separate Lattice/Vantis documentation.

Table 3-9. Place and Route - Quartus Description

Option	Description
Before this P&R tab is available you must load the Altera library, complete the design flow, and write an output netlist file. **Note:** The **Run PR** button for Quartus is available only when your target technology is Altera.	
■Run Quartus	Select this box if you want to run Quartus for your EDIF file.
❑Bring up Quartus GUI	In contrast to MAX+PLUS II, this selection gives you the additional step of setting up a project and completing the design compilation.
Path to Quartus executable:	Select and type your path name.
Write Output For:	■Back Annotated Timing Analysis
❑Simulation: Produce a netlist with VHDL or Veriolog from P&R tool for ModelSim or other simulator simulation. Generates simulation files and SDF 2.1; enables VHDL or Veriolog writers in Quartus.	❍VHDL ❍Verilog



Click **Run PR**

Note: LeonardoSpectrum supports mapping your design to APEX 20K. Depending on the options selected, mapping to WYSIWYG primitives is either done by LeonardoSpectrum or by Quartus. By default, LeonardoSpectrum does mapping to WYSIWYG primitives. Quartus is the new place and route software from Altera. The Altera APEX technology provides support for WYSIWYG device primitives.

Refer also to the Additional Instructions section at the end of Chapter 4 for more APEX 20K information.

Back Annotation Tab

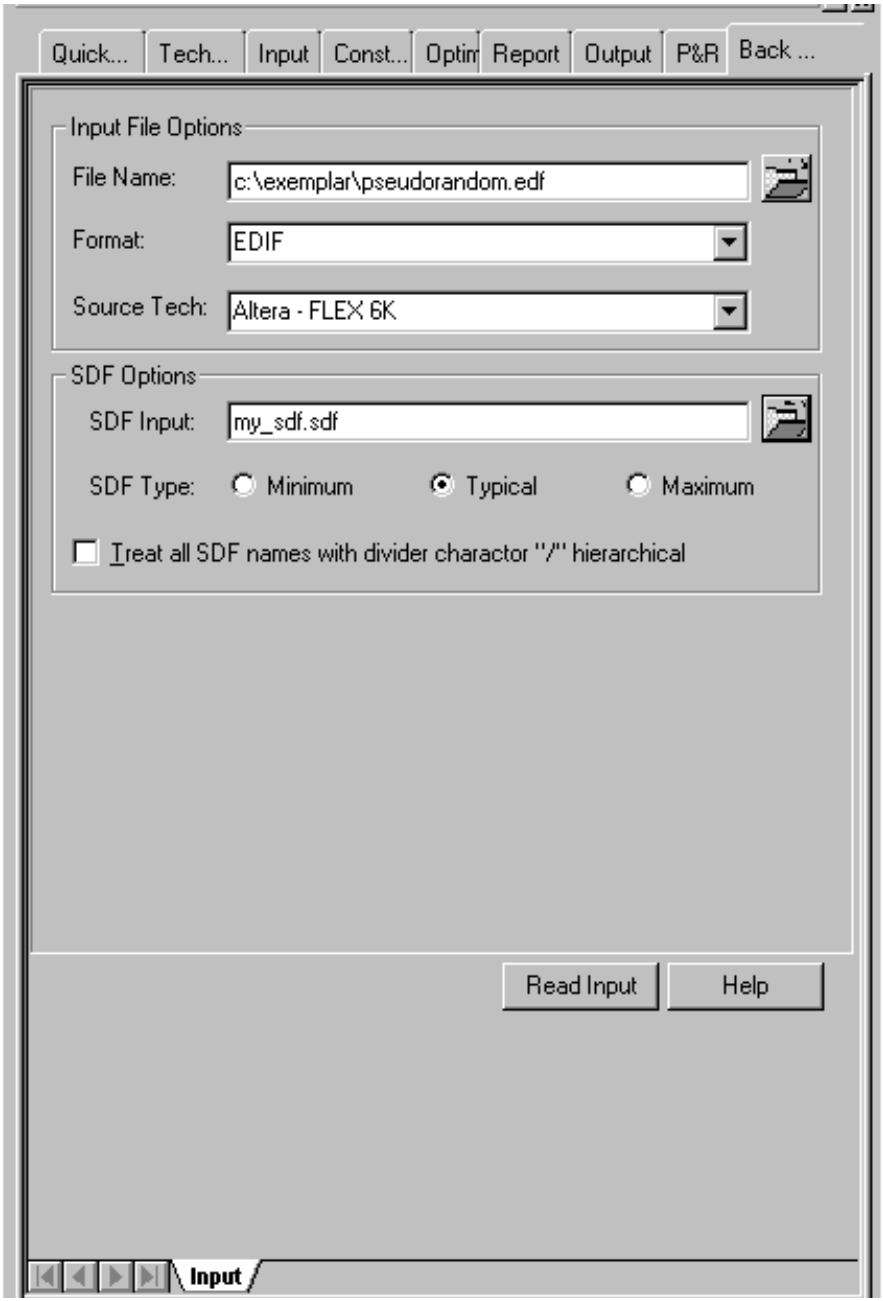
Table 3-10. Back Annotation - Input

Option	Description
Set options before reading in your back-annotated netlist and optional SDF file. The SDF file is used to produce a post place and route delay report.	
Input File Name	This file is an EDIF, VHDL, XNF, or Verilog file produced by vendor's software for place and route. Type or use pulldown.
Format	Specify the format of the input file in the format field. This file is used during timing analysis or back annotation. Type or use pulldown.
Source Tech	Specify the source technology, for example Altera FLEX, to be used during timing analysis or back annotation. This is the target technology for your design file and the location of your place-and-route netlist. **Note Xilinx:** Simprim and Neoprim library components are available on this list.
SDF Options	
SDF Input:	Specify the SDF input file to be read during back annotation. The vendor's place and route software produces this file.
SDF Type Specify delay derating for reading SDF file in LeonardoSpectrum.	❍Minimum ●Typical ❍Maximum
❑Treat all SDF names with divider character "/" hierarchical	The "/" character divides sub-modules within modules. You can differentiate between modules with the "/" character.
Click **Read Input**.	

After design synthesis, the generated netlist from LeonardoSpectrum can be run through place-and-route tools (P&RIntegrator) to generate a back annotation netlist.

Back annotation is the process of inserting actual delay numbers into the network after place-and-route. LeonardoSpectrum provides a mechanism for timing back annotation from the place-and-route tools. For most technologies, a separate Standard Delay Format (SDF) file is written by P&RIntegrator. **Note**: The Xilinx Alliance series, Altera APEX 20K Quartus, Altera FLEX MAX+PLUS II, and Lattice/Vantis Design Direct are encapsulated place-and-route environments in LeonardoSpectrum.

Screen 3-10. Back Annotation



Level 3 FlowTabs

4

FlowTabs are available after every startup. These tabs plus the power tabs bring more options to your design. The FlowTabs are designed for the advanced user who needs access to all the embedded power of LeonardoSpectrum. Nearly every step of the synthesis process can be customized based on FlowTabs. To use the FlowTabs you merely walk through each tab in order while customizing along the way. This is essentially what the SynthesisWizard and Quick Setup accomplish for you with default settings.

Note: Close Tip of the Day, if necessary. Click **Help** on FlowTabs for context assistance or click **Help** on menu bar for Help contents.

Synthesis FlowTabs

The FlowTabs and power tabs guide you through the synthesis flow process. This chapter is divided as follows:

- Quick Setup Tab, FPGA, ASIC
- Technology Tab, FPGA, ASIC
- Input Tab
- Constraints Tab
- Optimize Tab
- Report Tab
- Output Tab
- P&R Tab: Altera FLEX/Quartus, Xilinx, Lattice/Vantis
- Back Annotation Tab
- Additional Instructions

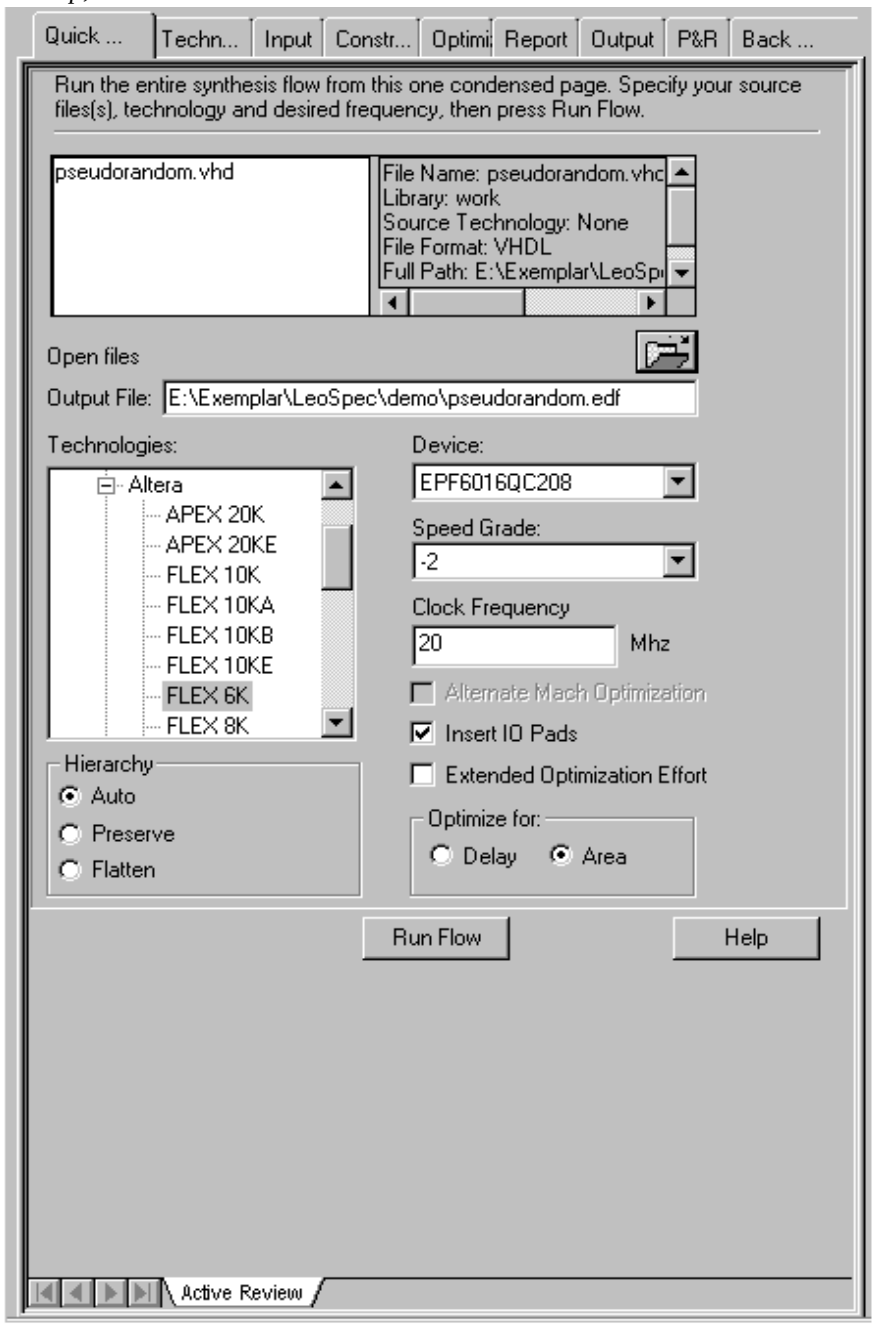
Quick Setup Tab - Active Review - FPGA

Quick Setup is intended for the user who is familiar with LeonardoSpectrum and the synthesis process. Everything that can be specified in the SynthesisWizard, can be specified on one condensed tab. Once specified, you can hit the Run Flow button to run the entire synthesis flow including synthesis, global constraints, optimization and writing netlist; and even set up retarget of an output netlist. In addition, Quick Setup automatically sets up all options, defaults, and settings in the FlowTabs to assist you when walking through the more advanced tabs. **Note:** Refer to Chapter 7 for adding a library to the GUI. Refer to Screen 4-1, Quick Setup FPGA and to SynthesisWizard Tutorial, Chapter 13. Active Review is a condensation of the SynthesisWizard. In addition, the options shown in Table 4-1 are also available on QuickSetup.

Table 4-1. Part of Quick Setup FPGA Options

Option	Description
Device:	LeonardoSpectrum selects a device. You can scroll to select another device.
Speed Grade:	Default speed grade (or process) for your device. Scroll to select another speed.
Clock Frequency	The recommended clock frequency is 20 MHz.
Alternate Mach Optimization	**Lattice/Vantis only**: Click to apply alternate synthesis heuristics. This flow may produce better results on small designs.
■Insert I/O Pads	If ■Insert I/O Pads is selected, then optimization runs in the chip mode and I/O pads are inserted in your design. If ❑ Insert I/O Pads is not selected, then optimization runs in the macro mode.
❑ Extended Optimization Effort:	If this box is selected, then LeonardoSpectrum runs additional optimization algorithms. Not selected by default.
Optimize for:	❍Delay - The design is faster and the area may be bigger. ●Area - The design is slower and the area may be smaller.
●Hierarchy Auto (default - auto_dissolve)	●Hierarchy Auto is selected by default. Views containing 3000 or fewer gates are dissolved.
❍Hierarchy Preserve	If ❍Hierarchy Preserve is not selected then your design is flatten (dissolved) before optimizing. If ●Hierarchy Preserve is selected then your design hierarchy is not changed during optimization.
❍Hierarchy Flatten (dissolve)	If ●Hierarchy Flatten selected then your entire design hierarchy is flattened.
Click **Run Flow** to start flow. **Note:** Run Flow is gray until you select a technology.	

Screen 4-1. Quick Setup, Active Review - FPGA



Quick Setup Tab - Active Review - ASIC

Quick Setup is intended for the user who is familiar with LeonardoSpectrum and the synthesis process. Everything that can be specified in the SynthesisWizard, can be specified on one condensed tab. In addition, Quick Setup automatically sets up all options, defaults, and settings in the FlowTabs to assist you when walking through the more advanced tabs. **Note**: Refer to Chapter 7 for adding an ASIC library to the GUI.

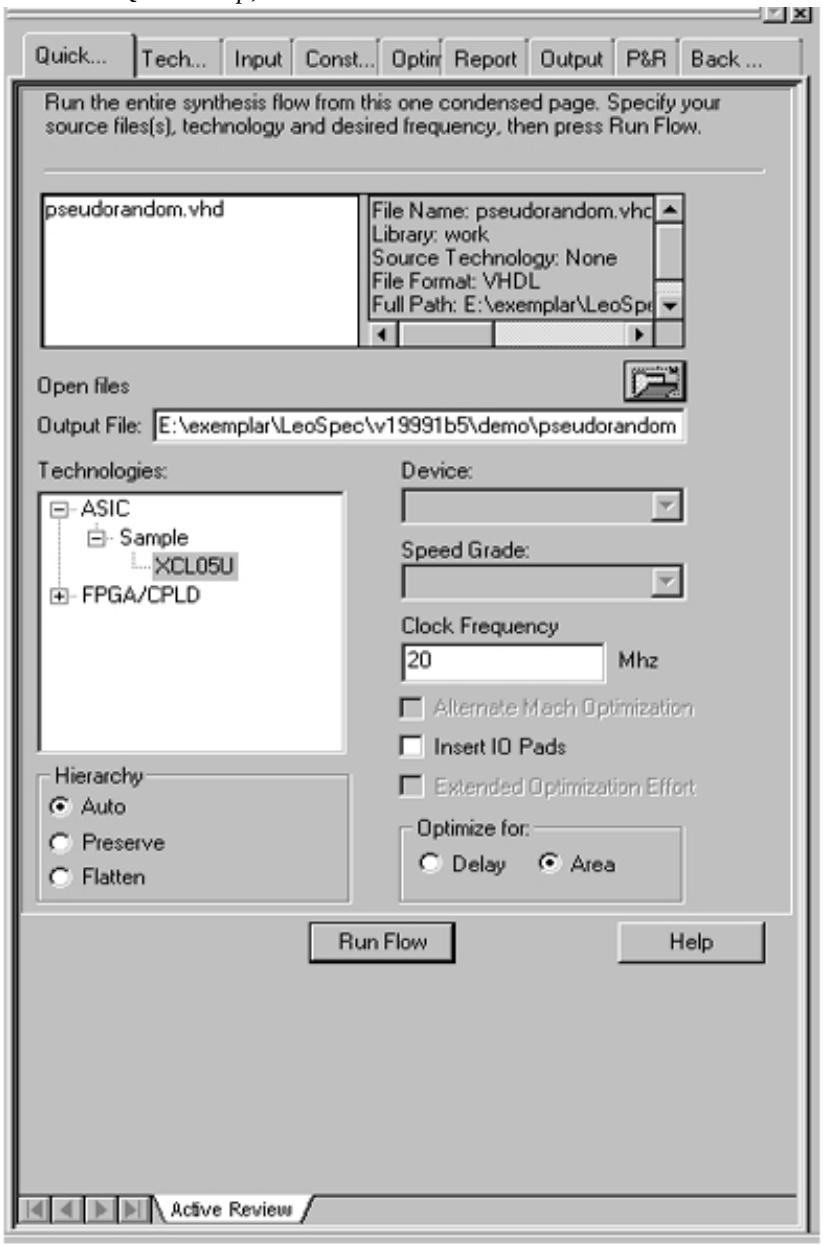
Note: If your technology is ASIC, then the ☐Extended Optimization Effort selection is grayed out, and the Device and Speed Grade scrollable fields are unavailable. However, technology independent optimization control can be done on the interactive command line shell. Refer to the `Optimize` command in the Reference guide for an explanation of the `-effort` option and the levels of effort (`quick`, `standard`, `remap`, `exhaustive`). Refer to Screen 4-2, Quick Setup, Active Review - ASIC, and refer to SynthesisWizard, Chapter 13. In addition to the SynthesisWizard setup, the options shown in Table 4-2 are also available on QuickSetup.

HP Platform Note: Refer to Additional Instructions in this chapter for an out-of-memory workaround that may be needed during optimization of a large design.

Table 4-2. Part of Quick Setup ASIC Options

Option	Description
Clock Frequency	The recommended clock frequency is 20 MHz.
Alternate Mach Optimization	**Lattice/Vantis** only: Click to apply alternate synthesis heuristics. This flow may produce better results on small designs..
■Insert I/O Pads	If ■Insert I/O Pads is selected, then optimization runs in the chip mode and I/O pads are inserted in your design. If ❑ Insert I/O Pads is not selected, then optimization runs in the macro mode.
Optimize for:	❍Delay - The design is faster and the area may be bigger. ●Area - The design is slower and the area may be smaller.
●Hierarchy Auto (default - auto_dissolve)	●Hierarchy Auto is selected by default. Views containing 30 or fewer gates are dissolved. If ❍Hierarchy Auto is not selected, then gates are not dissolved.
❍Hierarchy Preserve	If ❍Hierarchy Preserve is not selected then your design is flatten before optimizing. If ●Hierarchy Preserve is selected then your design hierarchy is not changed during optimization.
❍Hierarchy Flatten (dissolve)	If ●Hierarchy Flatten is selected then your entire design hierarchy is flattened.
Click **Run Flow** to start flow.	

Screen 4-2. Quick Setup, Active Review - ASIC



Technology Tab - FPGA

Refer to Screen 4-3, FPGA Technology Settings, and to Table 4-3. You can click on the technology icon to open a web browser.

Table 4-3. Technology Settings

Option	Description
Part:	This is the part number of your target device. LeonardoSpectrum selects the part number for you or you can select another from the pull down list (EPF6016QC208).
Speed:	This the speed grade of your target device. LeonardoSpectrum selects the speed (process) for you or you can select another from the pull down list (EPF6016QC208-2).

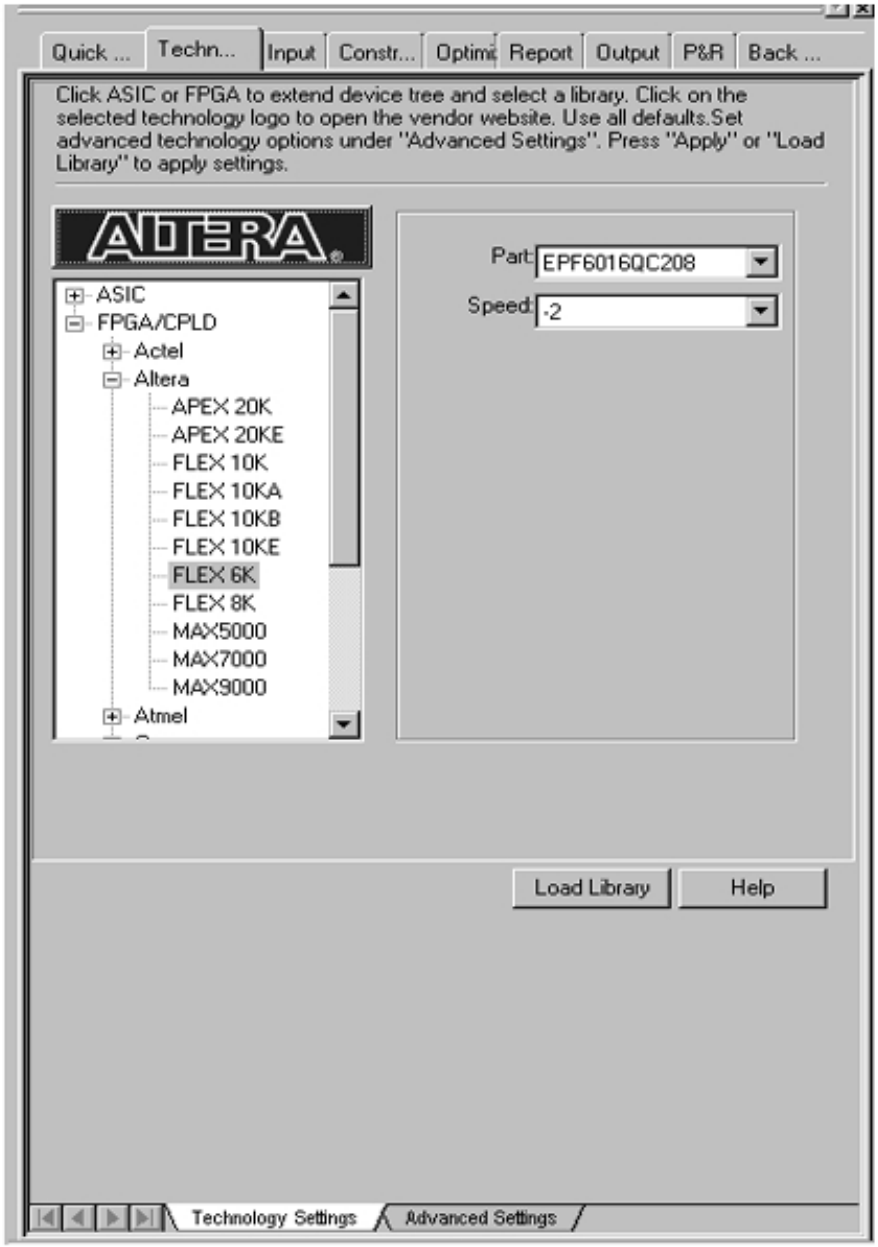
Click **Load Library** to load your technology library.

Note: If your format is XDB, then load the technology library before reading the input file. This sequence prevents problems with report delay and with the symbols in the schematic viewer.

Advanced Settings

Refer to Chapter 7 for Advanced Technology information.

Screen 4-3. Technology Settings - FPGA



Technology Tab - ASIC

Refer to Screen 4-4, ASIC Technology Settings and to Table 4-4 for a discussion of these settings.

Table 4-4. ASIC Technology Settings

Options	Description
Temperature:	Specify a temperature in celsius/centrigrade for use in delay calculations. This value overrides the default value in the target technology library.
Voltage:	Specify a voltage for use in delay calculations. This value overrides the default value in the target technology library.
Process:	Specify the process variation to be used in delay calculations. This value overrides the default value in the target technology library.

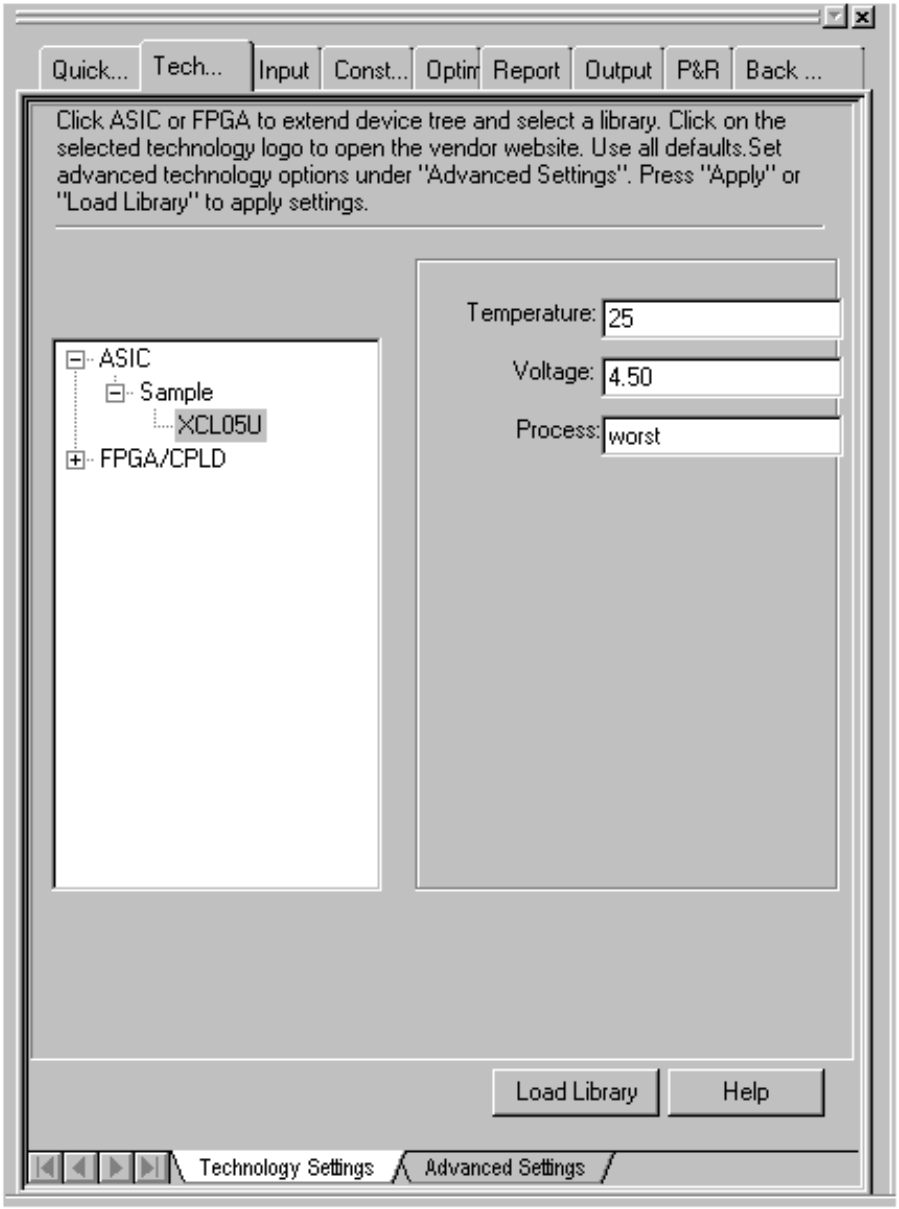
Click **Load Library** to load the technology library for your design.

Note: You can edit values for process, voltage, temperature (PVT) at any time on the interactive command line shell. Edit PVT on the fly before or after loading the library.

Note: There are approximately 35 ASIC technologies available. After you receive an ASIC package from an ASIC vendor, Exemplar, or Exemplar's website, continue to the **Adding a Library to the GUI** in Chapter 7.

Note: Refer to Chapter 7, Advanced ASIC Settings for information.

Screen 4-4. Technology Settings - ASIC



Input Tab

Refer to Screen 4-5, Input Files. This input tab is identical to the Input step for the SynthesisWizard in Chapter 13, except for the ☐ Analyze Only check box. Refer to Chapter 6, Batch Mode, for Encoding Style (`binary, onehot, twohot, random, gray, auto`) information. ■ Resource Sharing allows you to reduce number of devices. For example, you can replace two adders that have two inputs each with two muxes and one 2-input adder. **Note:** If ■ Analyze Only is selected, then your source code is checked for syntax errors before compiling; you must complete the Elaborate tab. **Rules:** Read, Analyze, Elaborate:

- Read does both analyze and elaborate functions.
- Read f1, read f2,....read f10. You can always read files. LeonardoSpectrum checks for accuracy of EACH file. This may take considerable time.
- Instead, you can analyze f1, analyze f2, ... analyze f10 and then elaborate. During elaborate, LeonardoSpectrum only needs to check accuracy once for all files. This may take less time than reading each file.
- You can also mix the functions by analyzing f1, then reading f2, and elaborating. You may use this if you want f1 to be a black box within f2 in your design, for example.

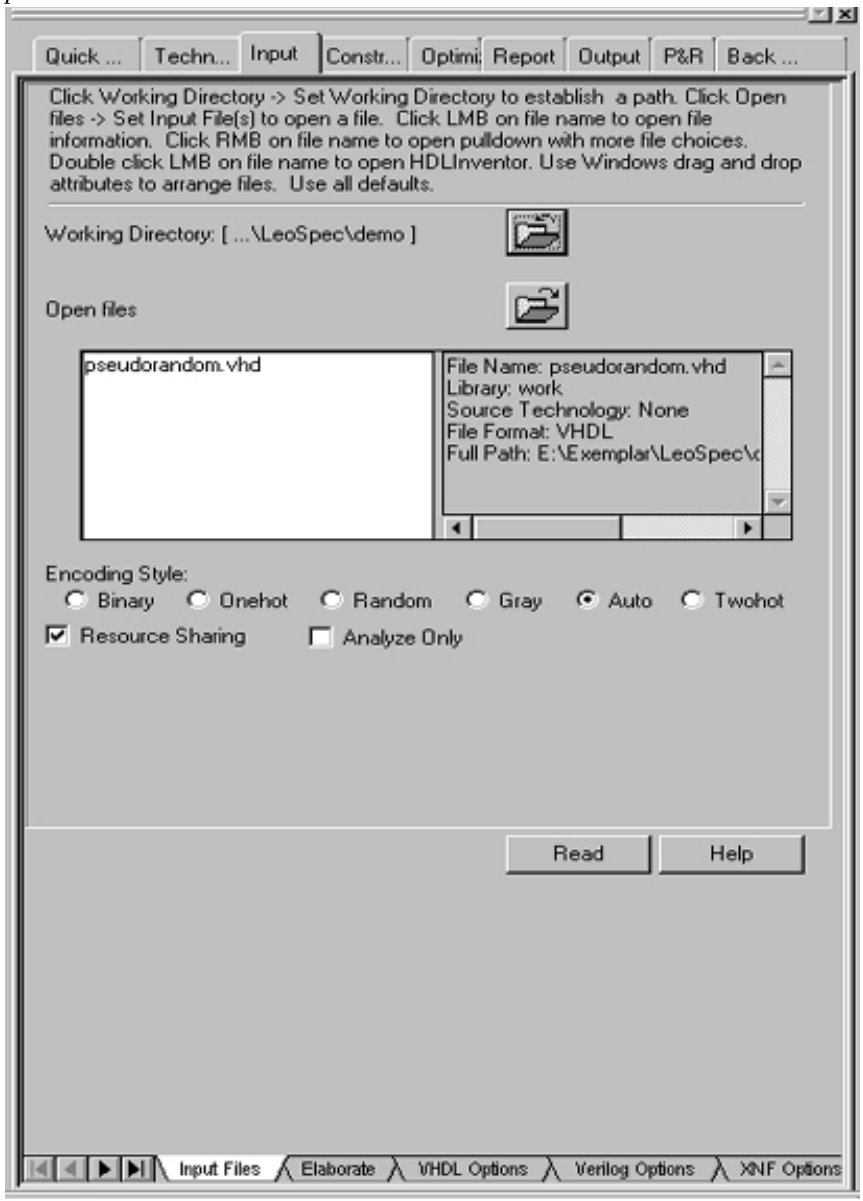
Refer to Chapter 7, Power Tabs and Advanced Topics for Elaborate, VHDL, Verilog, XNF, and EDIF information. Click **Read** to read your design into the database. **Note:** Refer also to Special Instructions for Mixing Design Languages in Chapter 7. **Note:** Double click LMB on input file to open the HDLInventor. **Note:** Use the Windows drag attributes to rearrange the input files, if necessary.

RMB over your input file to popup these shortcuts:

Add Input File: (Opens Set Input Files.)
Reverse Order
Toggle Selection
Select All
Open File: (Opens file in the Information Window.)
Set Work Library: (Opens Change work library.)
Set Technology (source): (Clears or Opens lists for FPGA, CPLD, and ASIC)
Set File Type: (Opens lists of output formats including XDB.)
Remove: (Click to remove highlighted input file.)
Remove All: (Click to clear the entire workspace.)

Note: Refer to Additional Instructions section for output netlist retarget steps.

Screen 4-5. Input Files



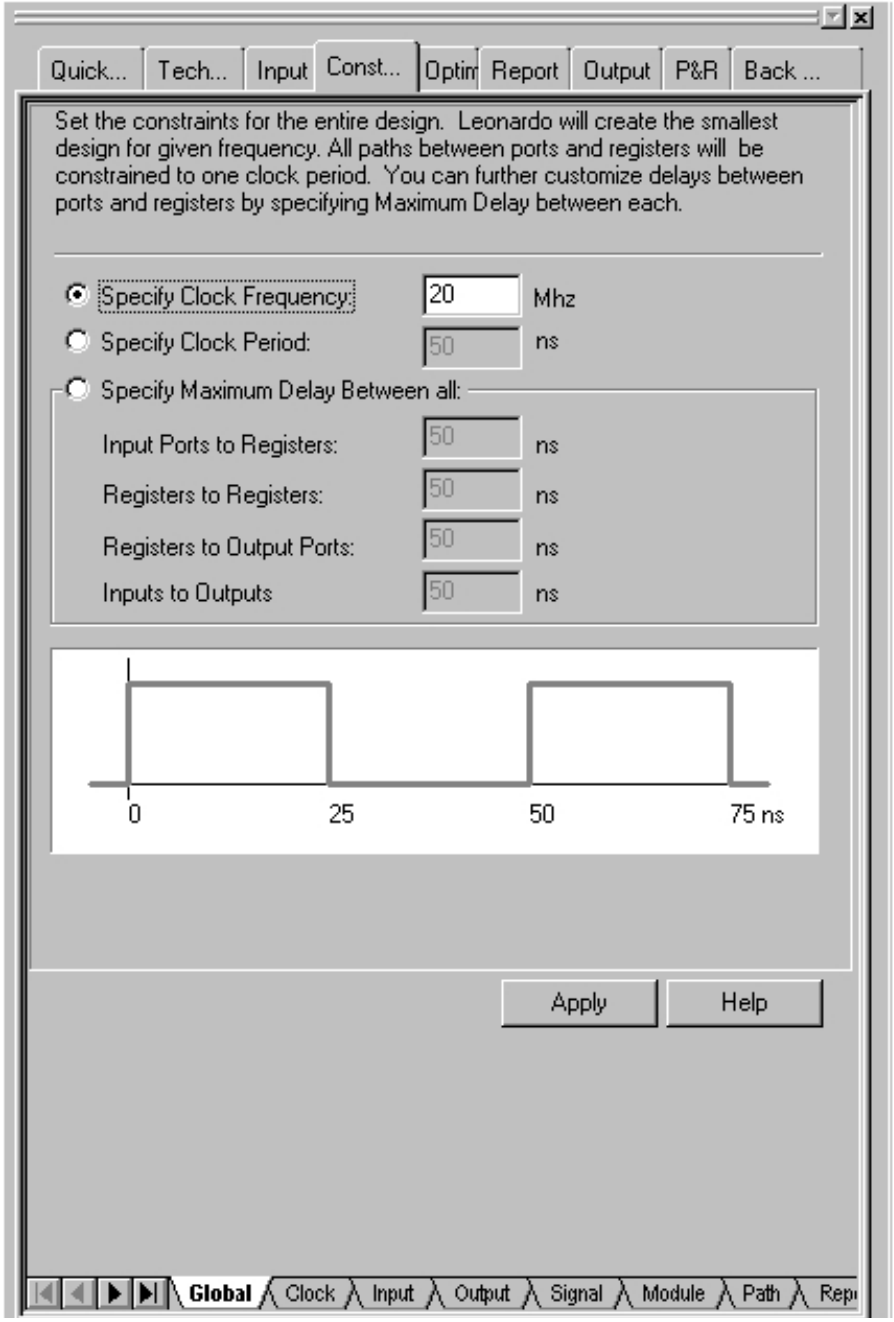
Constraints Tab

Refer to Screen 4-6, Global Constraints. The eight scrolling power tabs are explained in Chapter 5, Constraint Editor. Table 4-5 is duplicated here from Table 5-1, in Chapter 5, for convenience. Click **Apply** to apply options.

Table 4-5. Global Constraints

Option	Description
●Specify Clock Frequency, (Mhz):	This radio button choice is mutually exclusive with Clock Period. You can specify the required frequency for your design which is 1/period.
❍Specify Clock Period, (ns):	This radio button choice is mutually exclusive with Clock Frequency. See clock period in diagram.
❍Specify Maximum Delay Between all:	This radio button choice gives you control over the delays from port to register, register to register, register to port, and port to port.
Input Ports to Registers:	Delay from input port to input of register in nanoseconds.
Registers to Registers:	Delay from output of one register to input of another register in nanoseconds.
Registers to Output Ports:	Delay from output of register to output port in nanoseconds.
Inputs to Outputs	Delay from input port(s) to output port(s).
Waveform Window	The clock pulses show your settings.

Screen 4-6. Global Constraints



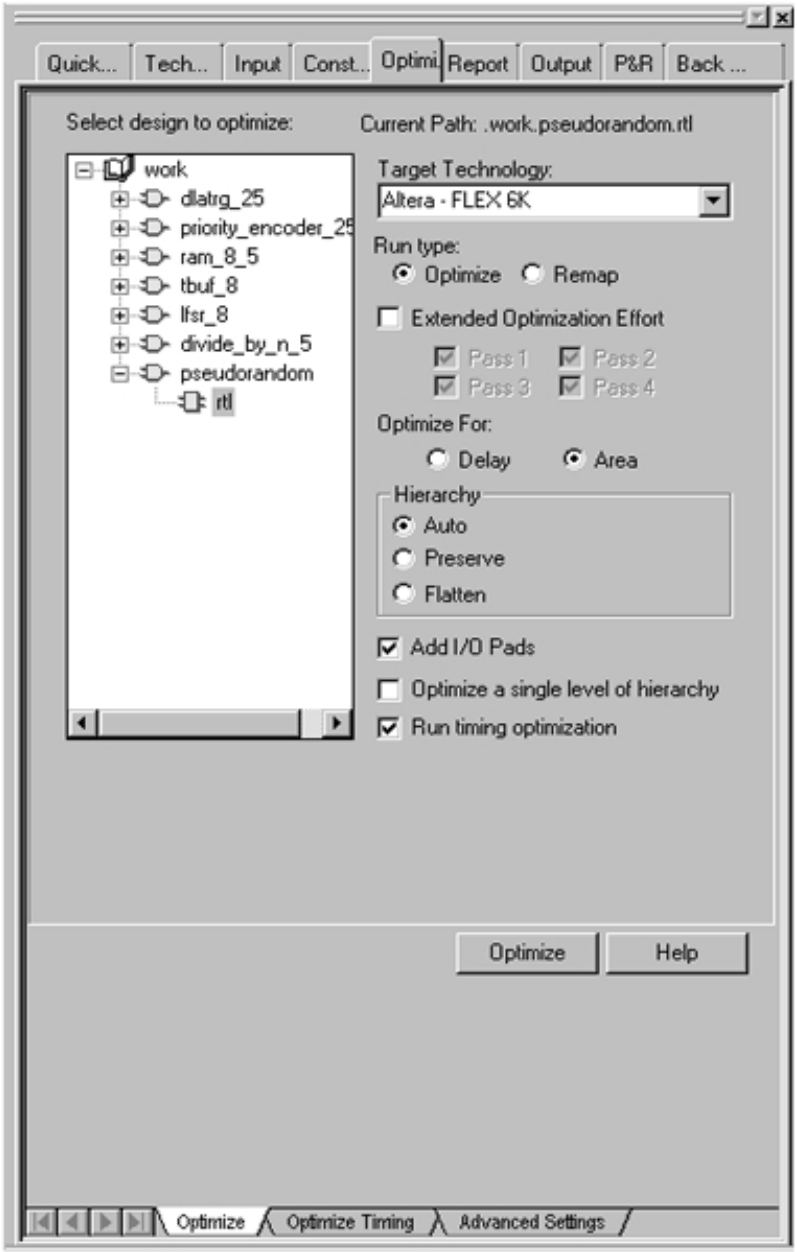
Optimize Tab

Refer to Screen 4-7 and to Table 4-6 for a discussion on optimize. Refer to Chapter 7, Table 7-10, for Optimize Options. **HP Platform Note:** Refer to Additional Instructions in this chapter for an out-of-memory workaround that may be needed during optimization of a large design.

Table 4-6. Optimize

Option	Description
Select design to optimize: Click to select an object from the filtered, embedded design browser tree.	
Current Path: This is the design you select from the design browser tree or your present design. Target Technology: Your current target is shown in the window.	
Run type: Control your run.	●Optimize: Default. Runs multiple optimization passes. Optimize means to reduce and improve logic in your design in terms of area and delay. ❍Remap: Does not optimize the network, but maps it into the target technology. The target may be another technology (retarget).
❑Extended Optimization Effort: If this box is selected, then LeonardoSpectrum runs an additional three optimization algorithms (assumes all 4 Pass boxes are selected). While selecting Passes (1-4) may cause a slower run, an improvement in the use of design space may occur. The Run type: ●Optimize: mustalso be selected.	
A report is made for:	❑ Pass 1 ❑ Pass 2 ❑ Pass 3 ❑ Pass 4
Optimize for:	❍Delay - The design is faster and the area may be bigger. ●Area - The design is slower and the area may be smaller.
●Hierarchy Auto (default - auto_dissolve)	●Hierarchy Auto is selected by default. Views with 3000 or fewer gates (CPLD/FPGA), or with 30 or fewer gates (ASIC) are dissolved.
❍Hierarchy Preserve	If ❍Hierarchy Preserve is not selected then your design is flatten before optimizing. If ●Hierarchy Preserve is selected then your design hierarchy is not changed during optimization.
❍Hierarchy Flatten	If ●Hierarchy Flatten is selected then your entire design hierarchy is flattened.
■Add I/O Pads	■Add I/O Pads is selected by default. LeonardoSpectrum runs the optimization in the chip mode and inserts I/O pads in your design. If ❑ Add I/O Pads is not selected, then LeonardoSpectrum runs the optimization in the macro mode.
❑Optimize a single level of hierarchy	If this box is selected then optimizing is limited to a single level (the current level) instead of all levels.
■Run timing optimization	The optimizations are concentrated on paths in the design that violate timing.
Click **Optimize** to optimize your design.	

Screen 4-7. Optimize



Report Tab

Refer to Screens 4-8 and 4-9 and to Tables 4-7 and 4-8. The task buttons are **Report Area** for Screen 4-9 and **Report Delay** for Screen 4-10. The power tabs are Report Area and Report Delay.

Table 4-7. Report Area

Options	Description
Report Filename:	Click to bring up Windows Set Report File or type in your report filename. The output appears in the LeonardoSpectrum transcript window.
■ Report Cell Usage	This is an option for the `report_area` command. `-cell_usage` option reports the total number of cells used for each instance in the entire design.
□ Report all levels of hierarchy	This is an option for the `report_area` command. `-hierarchy` option reports on all levels of hierarchy in your design on a module-by-module basis.
■ Report all leafs	This is an option for the `report_area` command. `-all_leafs` option reports only a particular module, level, and blackbox in your design.

Report Area: Click to generate an area report in the Information window.

Note: Use the `report_area -all_leafs` command for a technology-independent (unoptimized) design. The report includes the total number of primitives (AND, OR) and operators (add, subtract, multiply) and a count of the black boxes.

On a mapped (optimized) design, the `report_area -all_leafs` report includes technology-specific information: for example, function generators and flip-flops for Xilinx designs; and combinatorial and sequential modules for Actel designs.

- Arrival time is the time when a circuit node in a path changes state due to changes in design input.
- Required time is the time when this same node must no longer change to meet path constraints.

Screen 4-8. Report - Report Area.

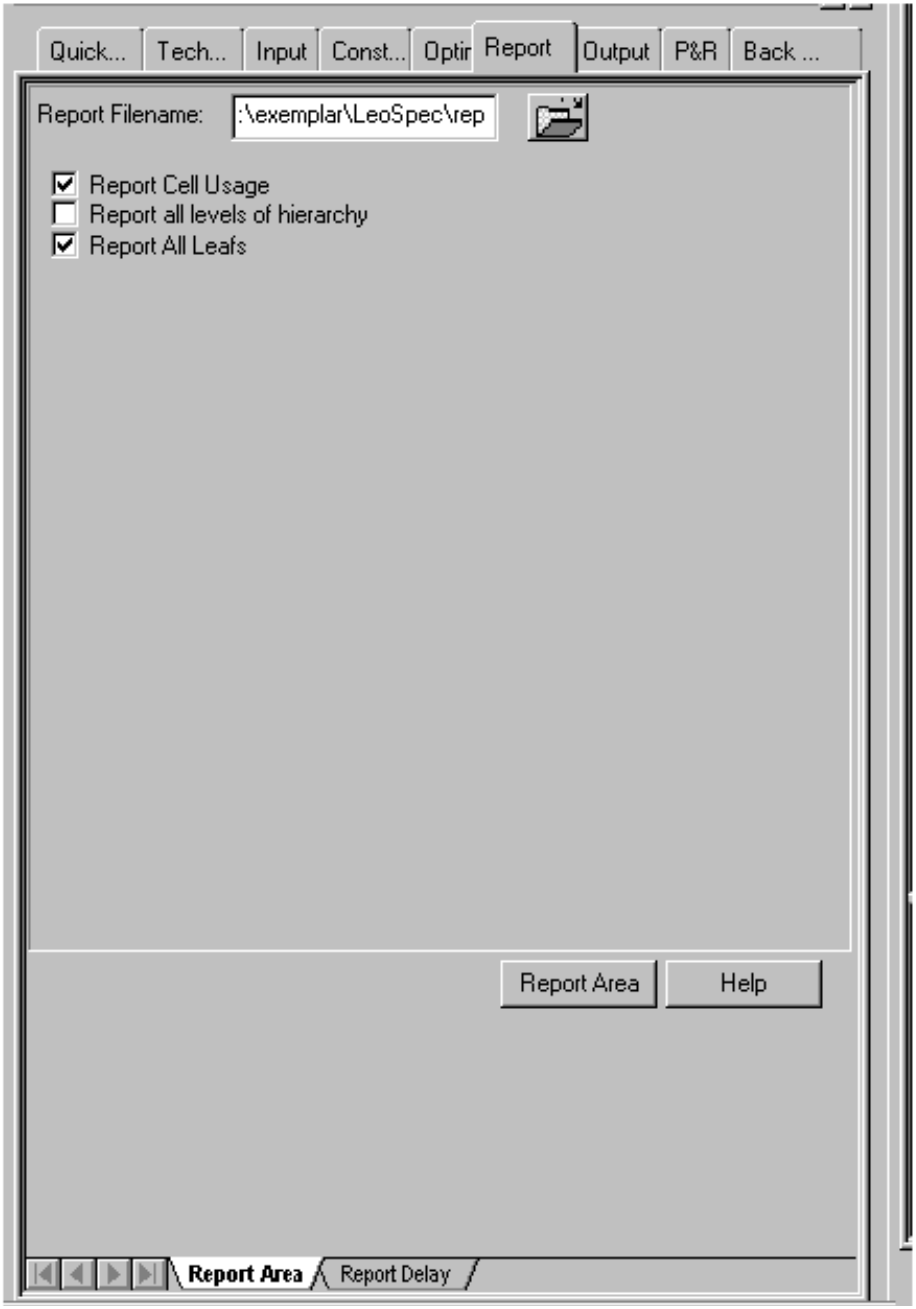
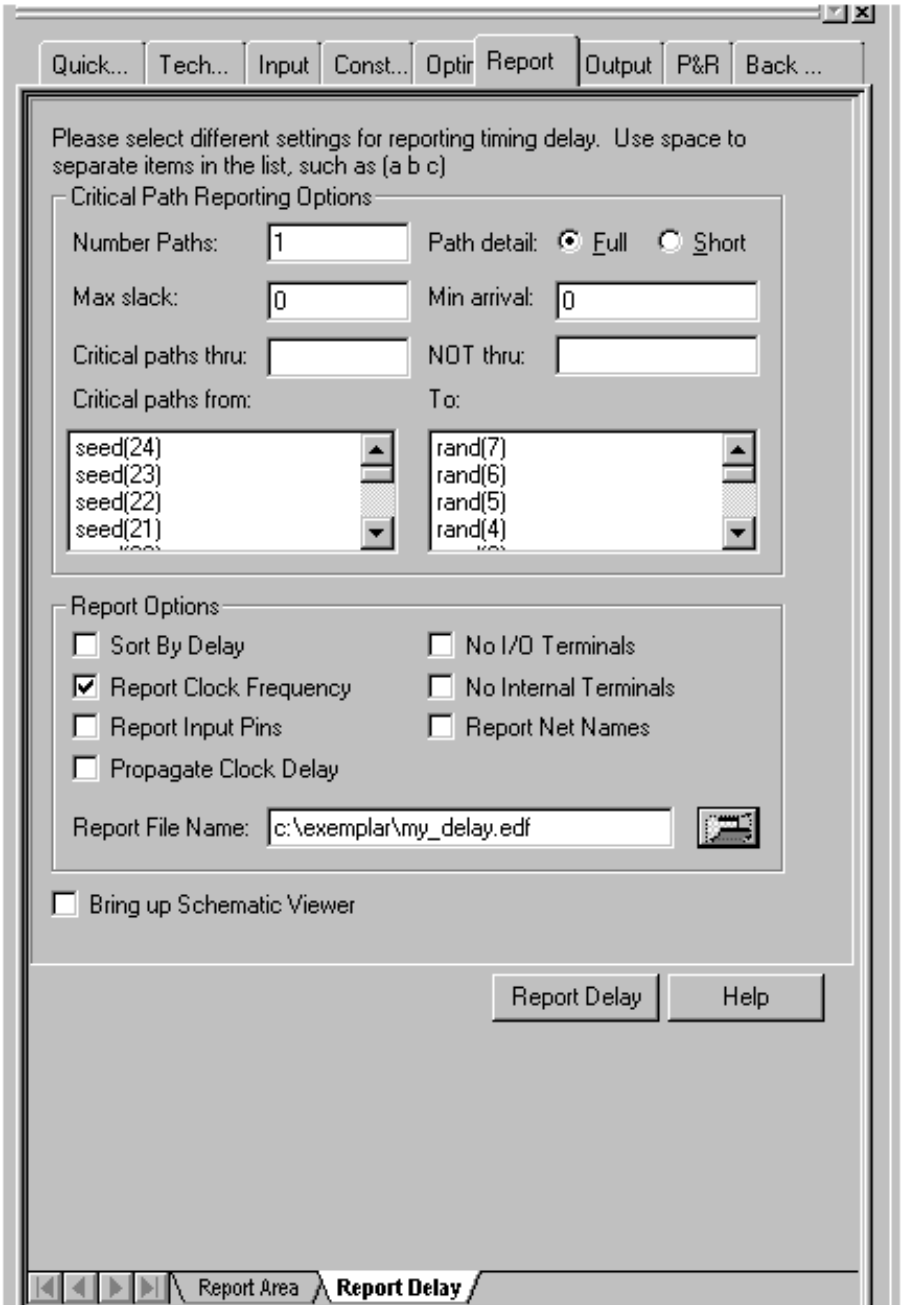


Table 4-8. Report Delay Options

Options	Description
Critical Path Reporting Options	
Number of Paths	By default the `report_delay` command reports on 1 path.
Path detail: ●Full ❍Short	**Full** detail shows point-to-point area, slack table, and 1 to n critical path reports. **Short** detail shows only an area report - path start and end point.
Max slack:	This is the number of constraints below threshold. Default threshold is 0.0.
Min arrival:	Arrival time is the time when a circuit node in a path changes state due to changes in design input. Arrival times greater than the min are reported.
Crit (critical) Paths Thru:	Filters and reports on paths through an object. Specify any number of points.
Crit Paths Not Thru:	Filters and reports on paths that do not pass through specified points.
Crit Paths From: (scroll window)	Filters and reports on paths starting at any number of points.
Crit Paths To:	Filters and reports on paths ending at any number of points.
☐ Sort by Delay	Report on the longest path first. Paths are sorted with latest arrival time first instead of by slack.
■ Report Clock Frequency	Compare actual with expected clock frequency, MHz.
☐ No I/O Terminals	Filters out paths that terminate in primary outputs.
☐ No Internal Terminals	Filters out paths that terminate at register inputs and blackboxes.
☐ Report Input Pins	When selected, reports on input pins; otherwise reports on output pins.
☐ Report Net Names	When selected, reports on net fanout in last column; otherwise reports on nets in critical path.
☐ Propagate Clock Delay	This choice is off to represent an ideal clock.
Report File Name: Click on 🗁 folder to bring up the Report File Name.	
☐ Bring up Schematic Viewer	The critical path for your design is displayed.
Report Delay: Click to generate a delay report.	

Note: These are options for the `report_delay` command. LeonardoSpectrum, uses slack analysis to evaluate every node in order to determine which paths are the most critical. Slack is the difference between the required (constraint) time and the arrival time (inputs and delays). Negative slack indicates that constraints have not been met, while positive slack indicates that constraints have been met. LeonardoSpectrum analyzes the circuit over a single clock cycle in terms of edge directions, differences between clock and data, and differences between sequential and combinational gates.

Screen 4-9. Report - Report Delay



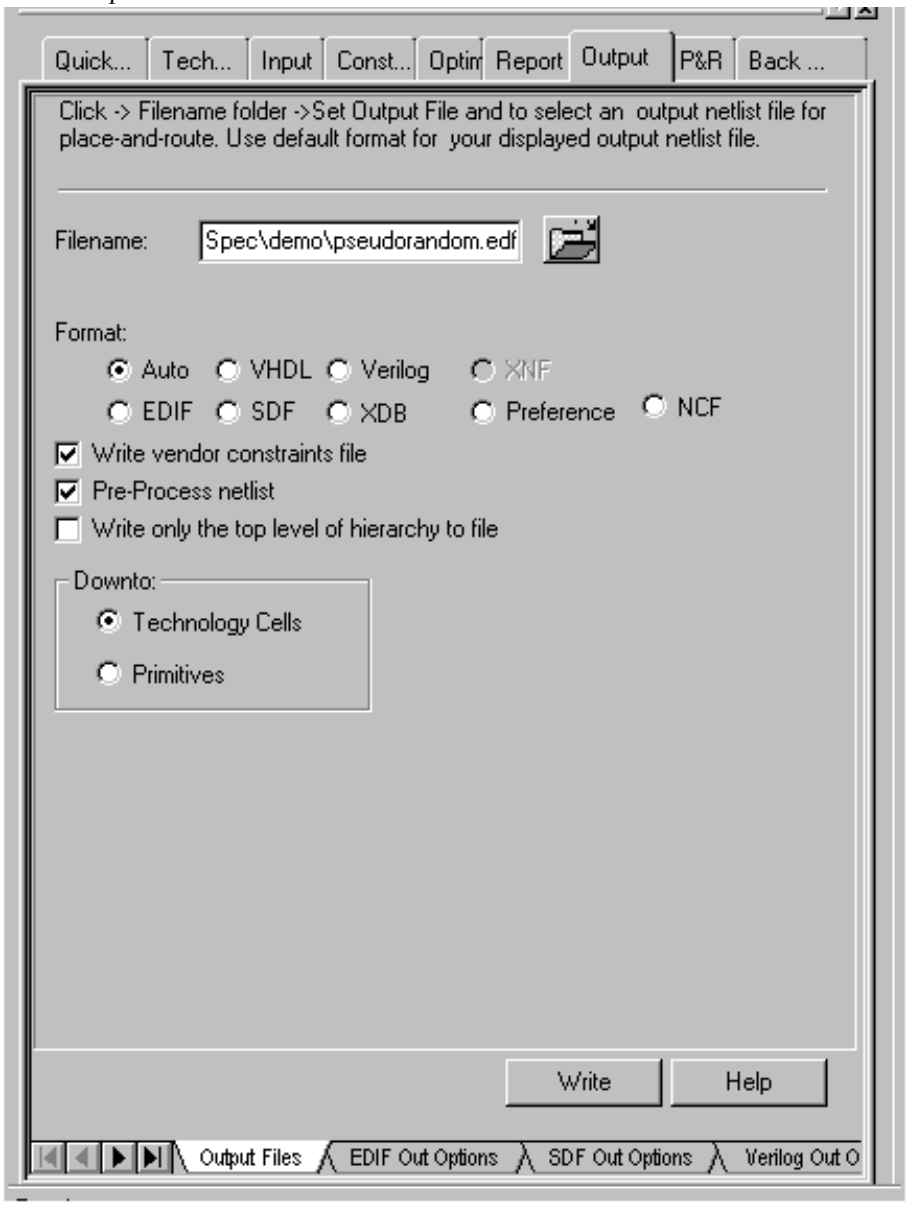
Output Tab

Refer to Screen 4-10 and to Table 4-9. Refer to Chapter 7, for power tab options.

Table 4-9. Output Files

Option	Description
🗁Filename: Note: Use a "-" for filename to have output appear on the main window.	Click on folder to bring up the Set Output Files. This is the place-and-route file. Select from list or type in another filename. This filename defaults to <input design>.<ext>, where <ext> is based on the output format. **Note:** Point at filename to popup the full path name.
Format:	The radio button output netlist format choices are listed.
●Auto	By default the output file is written out in EDIF format (.edf). Auto determines the actual format based on the filename extension.
❍VHDL	The output design file is in the VHDL netlist format.
❍Verilog	The output design file is in the Verilog netlist format.
❍XNF (Xilinx Netlist)	The output design file is in the XNF netlist format. Enabled only for Xilinx.
❍EDIF	The output design file is in the EDIF netlist format.
❍SDF (Standard Delay Format)	The output file is in the SDF netlist format. Output files in SDF are accepted by all technologies. This is a back-annotated SDF file.
❍XDB (Exemplar Database)	The output file is saved in a format that can be read back into LeonardoSpectrum without processing the netlist to remove technology-specific information. XDB writes a binary dump of your database to a file. You can read this file back into LeonardoSpectrum to restore the design database to the original conditions when the design was produced. **Note:** If your input format is XDB, then load the technology library before reading the input file. This sequence prevents problems with report delay and with symbols in the schematic viewer.
❍Preference (ORCA)	The output file is technology-specific for ORCA technologies.
❍NCF (Net Constraint File)	The output file contains technology-specific timing constraints.
■Write vendor constraints file: Select to write output file and a vendor's constraint file.	
■Pre-Process Netlist: By default executes auto_write command to write a netlist that meets requirements of your P&R tool. If ❒Pre-Process Netlist is not selected, then the write command is executed.	
❒Write only the top level of hierarchy to file: Select to limit output file to only the top hierarchy of file.	
Downto: (leaf-level)	●Technology Cells: Output file includes technology cells. ❍Primitives: Output file includes your original design (leaf level).
Click **Write** to apply your options.	

Screen 4-10. Output Files.



P&R Tab

Refer to Screens 4-11, 4-12, and 4-13; and to Tables 4-10, 4-11, 4-12, and 4-13.

Table 4-10. Place and Route - Altera Description

Option	Description
Before this P&R tab is available you must load the Altera library, complete the design flow, and write an output netlist file. **Note:** The **Run PR** button is available only when your target technology is Altera.	
■ Setup MAX+PLUS II Create (Assignment and Configuration File) ACF file: This option is selected by default. ❑ Setup MAX+PLUS II allows you to change or overwrite an existing ACF file.	
❑Auto Fast I/O	Select to allow MAX+PLUS II compiler to implement registers in Fast I/O. Reduces area requirements, but slows internal circuitry.
❑Auto Implement in EAB (Altera FLEX 10K)	Select this box if you are using wide gates and want to embed the array block.
❑Auto Register Packing	If your registers always have a constant input, for example “1”, then these registers are merged in the EDIF. Implements register packing by placing a combinational logic function and a register with a single data input in the same logic cell.
■Run MAX+PLUS II	Select this box if you want to run MAX+PLUS II for your EDIF file.
❑Bring up MAX+PLUS II	You want to bring up the MAX+PLUS II GUI.
■Timing Analysis	Use the timing information in the SDF, VHDL, or Verilog file to check place and route for accuracy. Creates either an input to output delay matrix, a setup/hold matrix, or a register performance report.
❍Input-Output Delay	Select for typical delay.
❍Setup/Hold	You want to check on setup and hold violations.
●Register Performance	You want to verify that constraints are met.
Path to MAX+PLUS II executable	This is the path to the Altera script🗁 to invoke MAX+PLUS II.
Write Output For:	■Back Annotated Timing Analysis
❑Simulation: Produce a netlist with VHDL or Verilog for Model Simulation or other simulator. Generate simulation files and SDF 2.1; enables MAX+PLUS II writers.	❍VHDL ❍Verilog
Click **Run PR** to create your ACF file and to invoke MAX+PLUS II using specified options.	

Screen 4-11. Place and Route - Altera.

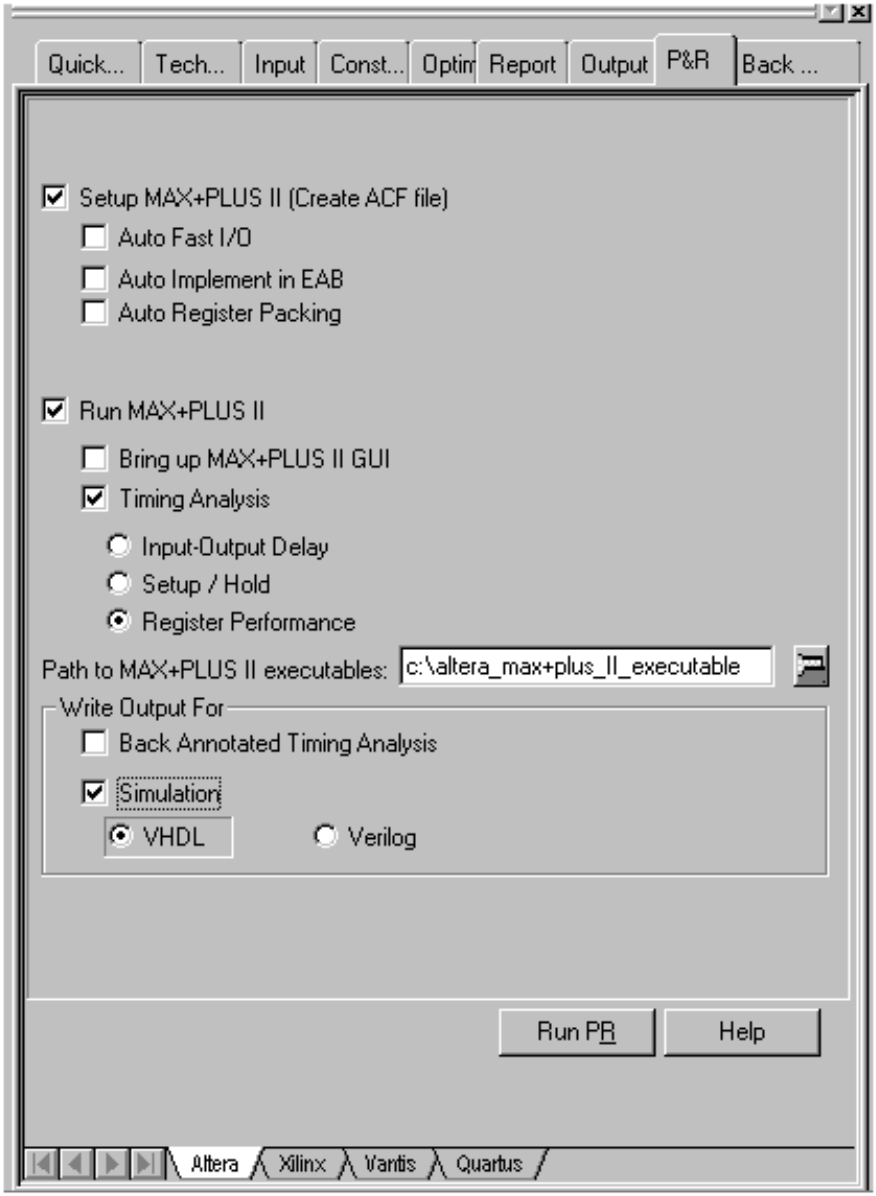


Table 4-11. Place and Route - Xilinx Description

Option	Description
Setup your place and route options and invoke the Xilinx M1 tools. Before this P&R tab is available you must load the Xilinx library, complete the design flow, and write an output netlist file.	
● Execute Place Route - Startup the DesignManager	
Effort	●Standard Uses standard effort to perform place and route. ❍High Performs place and route for better results than Standard, but takes a longer time to run.
❑ Generate Files for timing simulation	After the EDIF place and route, then run the SDF, VHDL or Verilog files through back annotation to simulate the router design.
❑ Generate bit file	Click to generate a bit file after place and route to program a Xilinx device.
❑ Use bitgen command file:	Click to bring up Bit Gen Command File 🗁or type in a file name. This file contains options for the Xilinx bit file generator.
❍Only generate netlist for functional simulation.	Click to only generate a VHDL or Verilog file. A SDF timing file is not needed.
❍Only generate pre place & route delay estimate	Click if you want to generate an estimate of place and route instead of actually doing a place and route.
❍Run Design Manager	Click if you want to run the Xilinx M1 GUI.
❑ Gate Level back annotation	Click if you want to do gate level with your EDIF file. Use an NGM file with ngdanno to perform gate level back annotation.
Install path for place_route exec:	This is the path to the 🗁directory where your Xilinx M1 executable is installed. If the Xilinx environment variable is set or if the executable is in your path then leave this field blank.
Write Output For:	■Back Annotated Timing Analysis
❑Simulation: When selected, LeonardoSpectrum instructs Alliance Series to generate VHDL or Verilog simulation netlists to target the simprim or neoprim primitive cell set, and to generate an SDF file for back annotated timing.	❍VHDL ❍Verilog
Click **Run PR** to invoke Xilinx M1 tools using specified options.	

Screen 4-12. Place and Route - Xilinx

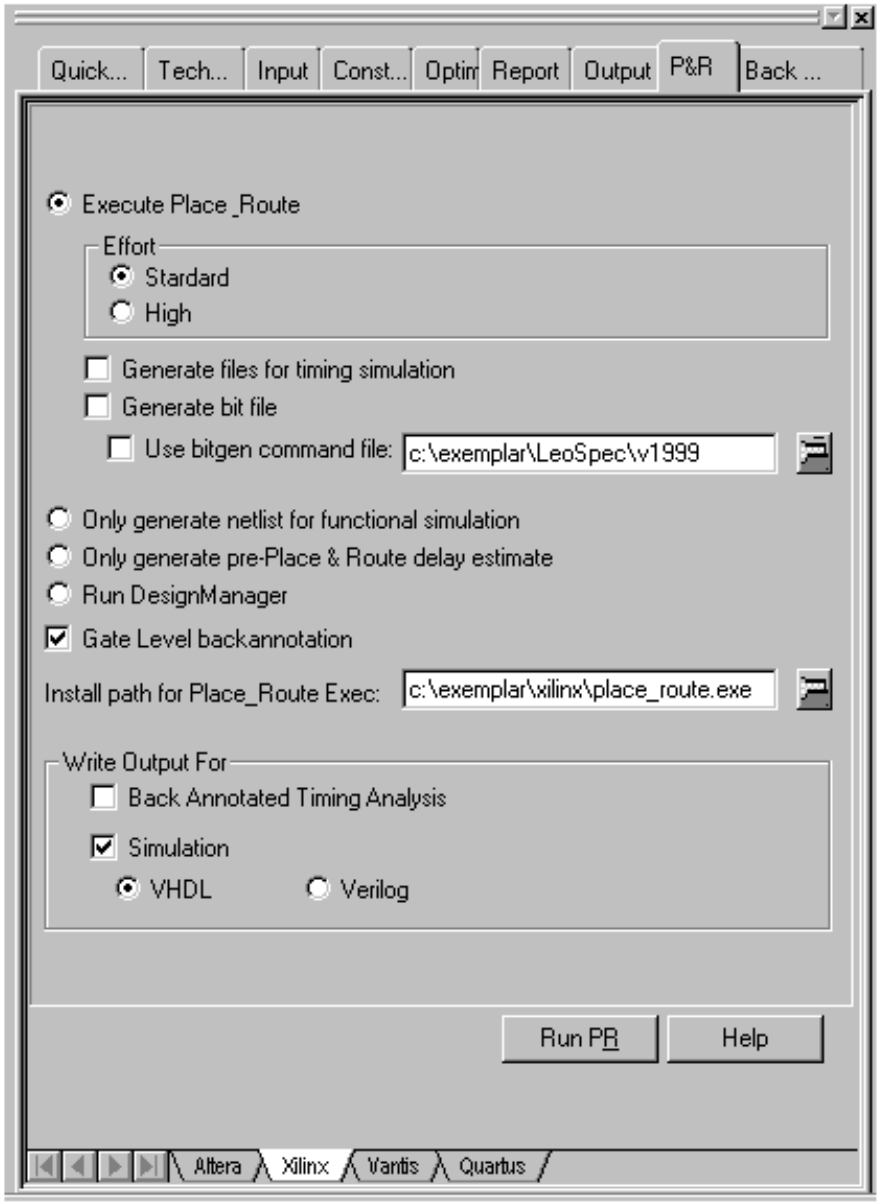
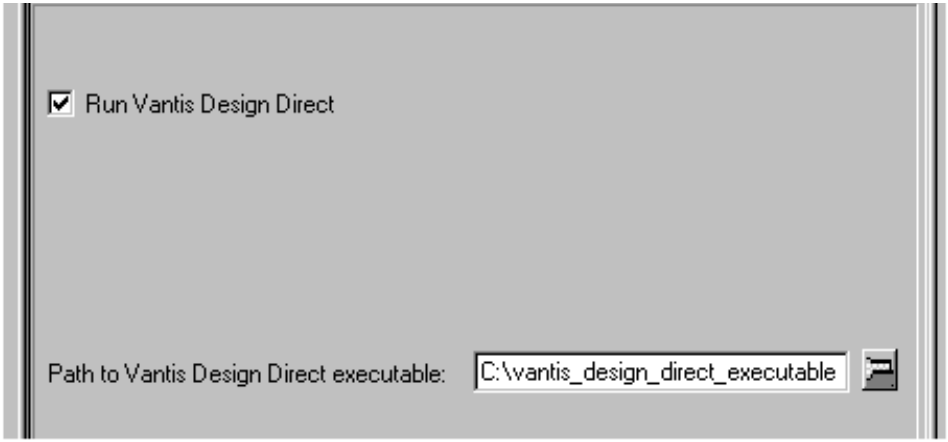


*Table 4-12. Place and Route -Lattice/ Vantis**

Option	Description
■ Run Lattice/Vantis Design Direct	Run the Lattice/Vantis Design Direct GUI.
Type a path for your Design Direct executables.	

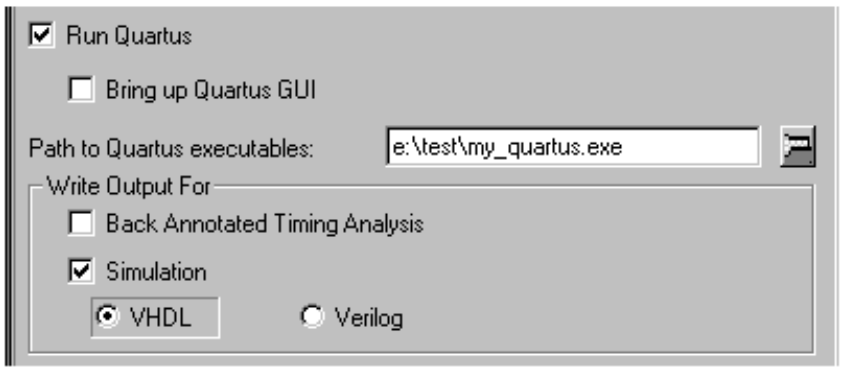
Screen 4-13. Place and Route - Lattice/Vantis



*Refer also to separate Lattice/Vantis documentation.

Table 4-13. Place and Route - Quartus Description

Option	Description
Before this P&R tab is available you must load the Altera library, complete the design flow, and write an output netlist file. **Note:** The **Run PR** button is available only when your target technology is Altera.	
■Run Quartus	Select this box if you want to run Quartus for your EDIF file.
❑Bring up Quartus GUI	In contrast to MAX+PLUS II, this selection gives you the additional step of setting up a project and completing the design compilation.
Path to Quartus executable:	Select and type your path name.
Write Output For:	■Back Annotated Timing Analysis
❑Simulation: Produce a netlist with VHDL or Veriolog from P&R tool for ModelSim or other simulator simulation. Generates simulation files and SDF 2.1; enables VHDL or Veriolog writers in Quartus.	❍VHDL ❍Verilog



Click **Run PR**

Note: LeonardoSpectrum supports mapping your design to APEX 20K/20KE. Depending on the options selected, mapping to WYSIWYG primitives is either done by LeonardoSpectrum or by Quartus. By default, LeonardoSpectrum does mapping to WYSIWYG primitives. Quartus is the new place and route software from Altera. The Altera APEX technology provides support for WYSIWYG device primitives.

Refer also to **Additional Instructions** at the end of this chapter.

Back Annotation Tab

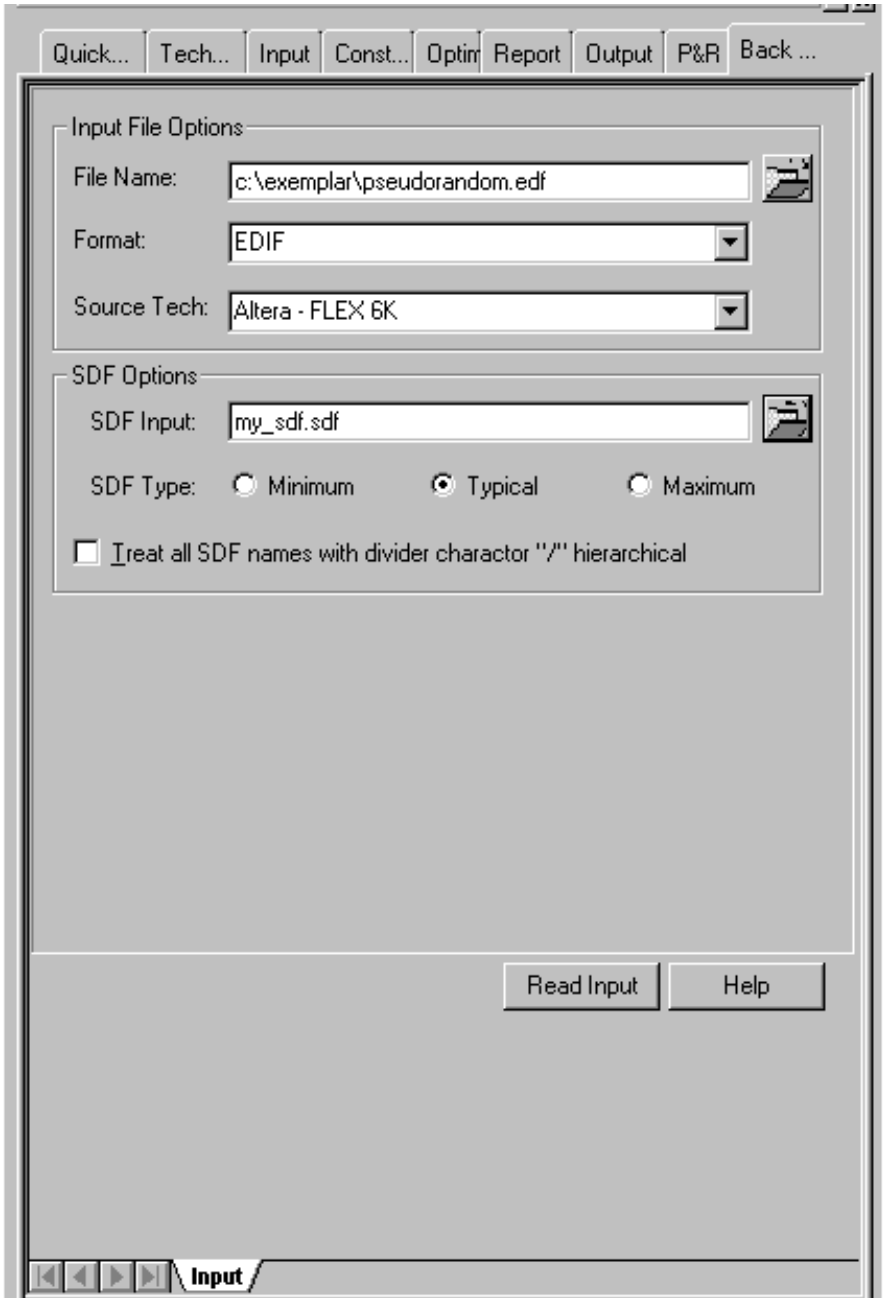
Table 4-14. Back Annotation - Input

Option	Description
Set options before reading in your back-annotated netlist and optional SDF file. The SDF file is used to produce a post place and route delay report.	
Input File Name	This file is an EDIF, VHDL, XNF, or Verilog file produced by vendor's software for place and route. Type or use pulldown.
Format	Specify the format of the input file in the format field. This file is used during timing analysis or back annotation. Type or use pulldown.
Source Tech	Specify the source technology, for example Altera FLEX, to be used during timing analysis or back annotation. This is the target technology for your design file and the location of your place-and-route netlist. **Note Xilinx**: The Simprim and Neoprim library components are available on this list. Simprim and Neoprim are also available when you RMB over the input file name.
SDF Options	
SDF Input:	Specify the SDF input file to be read during back annotation. The vendor's place and route software produces this file.
SDF Type Specify delay derating for reading SDF file in LeonardoSpectrum.	❍Minimum ●Typical ❍Maximum
❑Treat all SDF names with divider character "/" hierarchical	The "/" character divides sub-modules within modules. You can differentiate between modules with the "/" character.
Click **Read Input**.	

After design synthesis, the generated netlist from LeonardoSpectrum can be run through place-and-route tools (P&RIntegrator) to generate a back annotation netlist.

Back annotation is the process of inserting actual delay numbers into the network after place-and-route. LeonardoSpectrum provides a mechanism for timing back annotation from the place-and-route tools. For most technologies, a separate Standard Delay Format (SDF) file is written by P&RIntegrator. **Note**: The Xilinx Alliance series, Altera APEX 20K Quartus, Altera FLEX MAX+PLUS II, and Lattice/Vantis Design Direct are encapsulated place-and-route environments in LeonardoSpectrum.

Screen 4-14. Back Annotation



Additional Instructions

The following instructions are steps for:

- Synthesis Steps for Output Netlist
- Retarget Steps for Output Netlist
- Steps for auto_dissolve
- HP Platform: Out-of-Memory Workaround
- Hints for Back Annotation
- APEX 20K/20KE Variable Information
- Simpler Clock Enable Logic

Synthesis Steps for Output Netlist

Use these Quick Setup FlowTab steps to synthesize a netlist for your technology.

1. On Quick Setup, run your design flow to generate an output netlist file with an `.edf` extension.
   For example: Choose Altera FLEX 6K, 20 MHz, pseudorandom.vhd demo file.
   Note: On Quick Setup
   Output File is: `e:\exemplar\leospec\demo\pseudorandom.edf`
   Input File is: `pseudorandom.vhd`

2. LMB over `pseudorandom.vhd` to open file information into right window -> Source Technology: None.

3. Click **Run Flow**. When run is complete, open schematic viewer and verify that cells are Altera FLEX 6K.

4. Next, click RMB on `pseudorandom.vhd` input file to open list. Click Remove to remove `pseudorandom.vhd` and `e:\exemplar\leospec\demo\pseudorandom.edf` from Quick Setup.

5. You are now ready to retarget your output netlist to another technology. Continue to the next section, Retarget Steps for Output Netlist.

Retarget Steps for Output Netlist

Use these Quick Setup FlowTab steps to retarget a synthesized netlist from one technology to another technology. These steps assume you have completed the steps in the previous section, Synthesis Steps for Output Netlist.

1. When input and output file windows are blank, click Open Files -> Set Input File(s) -> `pseudonrandom.edf` to open your output netlist in the Input file window. **Note:** On Set Input File(s), choose Files of type -> All files *.* to locate .edf files.

2. On Quick Setup
   Output file is: `e:\exemplar\leospec\demo\pseudonrandom_1.edf`
   Note: LeonardoSpectrum automatically adds a "_1...n" extension to your retarget output netlist filename.
   Input file is: `pseudorando.edf`

3. Click RMB on `pseudonrandom.edf` to open list:
   Set Technology -> FPGA/CPLD -> Altera FLEX 6K. Click Altera FLEX 6K.

4. Click LMB on `pseudonrandom.edf` to bring file information into right window -> Source Technology: Altera FLEX 6K appears.

5. On Quick Setup Technologies: scroll to your target technology. For example, click on Xilinx 4000. Defaults change to Xilinx 4000.

6. Click **Run Flow**. When run is complete, open schematic viewer and verify that cells are now Xilinx 4000.

Steps for auto_dissolve

The hierarchy manipulation option is available from the GUI, interactive command line shell, and batch mode.

- If auto (auto_dissolve) is selected, then logic is equated to 2-input NAND gates, and hierarchy is dissolved according to the following rules:

(1) FPGA/CPLD auto_dissolve limit is 3000 gates (default).

(2) ASIC auto_dissolve limit is 30 gates (default).

(3) There is a maximum system limit of gates that can be dissolved in a module. This system limit cannot be modified by a user switch. If this limit is exceeded then auto_dissolve is not completed. The auto_dissolve dissolves instances in a context sensitive manner. If a module is instantiated more than once, then the instance is dissolved only if total number of gates does not exceed the system limit.

- If preserve is selected, then hierarchy is not changed during optimization.
- If flatten is selected, then the design hierarchy is flattened (dissolved).

Auto Dissolve Variables and Attribute:

The auto_dissolve_limit (FPGA/CPLD) and asic_auto_dissolve_limit (ASIC) variables, and the auto_dissolve attribute are available to dissolve blocks using the above three auto_dissolve rules. If the auto_dissolve attribute is set on an instance, then only that instance is dissolved. If the auto_dissolve attribute is set on a view, then all instantiations are dissolved. In addition, auto_dissolve is now the default for the optimization command in both the interactive command line shell and in batch mode. Hierarchy can be preserved with `-hierarchy <auto|preserve|flatten>` option on the interactive command line shell and in batch mode.

Note: Refer to Batch Mode chapter in this guide; and the Command Reference Guide, Commands, Variables, and Attributes chapters.

HP Platform: Out-of-Memory Workaround

Use these steps to reduce unused memory of the session.

1. After reading in the design, and before starting optimization, issue the following command:

   ```
   remove -hdl
   ```

2. Check and increase data size limitation of the workstation. Issue the following system command:

   Limits

cpu time:	unlimited
file size:	unlimited
data size:	2097148 kbytes
stack size:	8192 kbytes
core dump size:	0 kbytes
descriptors:	64
memory size:	unlimited

Note: If the data size is less than available swap memory, request the system administrator to change the kernel configuration and increase the data size.

Back Annotation Neoprim and Simprim Hints

After creating an output netlist in LeonardoSpectrum, the P&R tool for Xilinx may contain the Simprim or Neoprim library components. For back annotation you need to target the Xilinx Simprim or Neoprim library components. Select Simprim or Neoprim on the back annotation tab. **Note:** You can also LMB over the input file name on Quick Setup tab and select Neoprim or Simprim from pull down menu.

APEX 20K/20KE Variables and Other APEX 20K/20KE Information

LeonardoSpectrum supports mapping your design to APEX 20K/20KE. Depending on the options selected, mapping to WYSISYG primitives is either done by LeonardoSpectrum or by Quartus. By default, LeonardoSpectrum does mapping to WYSIWYG primitives. Quartus is the new place and route software from Altera. The Altera APEX technology provides support for WYSIWYG device primitives.

Mapping Options

By default, mapping to WYSIWYG primitives is `true` and mapping to complex I/Os is `false`. Currently, these options are not available in the GUI.

The variable `apex_wysiwyg_support` enables mapping to WYSIWYG ATOMs (an ATOM is a WYSIWYG primitive) in APEX 20K/20KE devices. If you need to turn this variable off, you must also set `dont_lock_lcells` to `true`. For example,

```
set apex_wysiwyg_support false
set dont_lock_lcells true
```

The variable `apex_map_complex_ios` enables mapping to I/O flip flops. This variable is functional if the variable `apex_wysiwyg_support` is also set to `true`. For example,

```
set apex_map_complex_ios true
set apex_wysiwyg_support true
```

APEX 20K/20KE Mapping

LeonardoSpectrum support for Mapping to Altera APEX 20K/20KE WYSIWYG cells includes:

1. By default, LeonardoSpectrum maps to all modes of Lcells:
   - Counters
   - QFBK_Counters
   - Arithmetic
   - Normal
2. Mapping to I/Os, including various complex I/O configurations is available.
3. RAMs/ROMS are now mapped to LPM_RAMs and LPM_ROMs. In the future, direct mapping to APEX 20K/20KE WYSIWYG primitives - RAM slices and Pterms (product terms) - will be provided by LeonardoSpectrum. LPM_RAMs and LPM_ROMs will implemented as RAM slices by Quartus.
4. APEX 20K/20KE libraries are supported.
5. The current P&R GUI for Quartus provides support for the P&R flow using the Quartus NativeLink API features. This allows you to access and modify designs in the Quartus database. Currently, the EDIF format is supported for the output netlist. In contrast to MAX+PLUS II, the choice box selection of "Bring up the Quartus GUI" allows the additional step of setting up a project and completing the design compilation.
6. Wireload model support is functional.
7. When possible, LeonardoSpectrum supports absorption of NOT gate into WYSIWYG primitives.
8. By default, GND/VCC are exported as cells. In Quartus, the preference is to export GND/VCC as undriven nets.

   Set the output->edif out->write power/ground as undriven nets to `true` in LeonardoSpectrum. For example, script command:
   ```
   set edifout_power_ground_style_is_net TRUE
   ```

More Quartus Information

Quartus for Altera APEX 20K/20KE: LeonardoSpectrum provides support for cross probing from within Quartus into the original HDL files.

When targeting APEX 20K/20KE, LeonardoSpectrum generates a cross reference .xrf file together with the EDIF netlist. This allows Quartus users to seamlessly crossprobe into the original HDL design files from the floor plan view.

Quartus for Altera APEX 20K/20KE: LeonardoSpectrum supports passing constraints to Quartus using the Quartus NativeLink API features.

Simpler Clock Enable Logic

Simpler clock enable logic is generated in many cases. For the following example, `en1` is now used directly as the clock enable. In previous releases, "`en1 AND en2`" was used.

```
process (clk) begin
    if rising_edge (clk) then
       if en1 = '1' then
          if en2 = '1' then
             z <= a;
          end if;
       end if;
    end if;
end process;
```

Constraints 5

LeonardoSpectrum allows you to control the optimization and mapping process with constraints. Constraints are entered in the constraint editor and then saved in a constraint file. Constraints can also be entered on the interactive command line shell (**Level 3**). This chapter is divided as follows:

- Constraint File
- Syntax
- Alphabetical List
- Attributes
- Low Level Block Constraints
- Constraint File Editor

Constraint File

LeonardoSpectrum automatically reads a default constraint file with the name `<input_filename>.ctr` which can be edited on the constraint editor report. If a different name is desired for the constraint file, then use the option: `-control=<name>`. Specify `-nocontrol` if the file with the default name is not required. Refer to the Constraint Editor section in this chapter. LeonardoSpectrum allows you to set and remove constraints on the constraint editor. Constraints can also be set or removed on the interactive command line shell (**Level 3**), for example:

```
set_attribute -port clk -name CLOCK_CYCLE -value 10
remove_attribute -port clk -name CLOCK_CYCLE
```

Syntax

Syntax rules and examples are presented for the following:

- Constraint File Command
- Syntax for Busses

Constraint File Commands

The constraint file consists of commands selected from a list. Each command is on a single line. The “#” character is a comment character. Everything on a line past “#” is ignored. If a command and comment are on the same line, then use a semicolon (“;”) as a separator.

The `set_attribute` command sets an attribute on object(s).

The `remove_attribute` command removes an attribute from object(s) and must be used for removing all attributes except for the `connect` and `disconnect` commands. `remove_attribute` provides the same function as the **Delete Constraints** button on the constraint editor.

You can list attribute(s) on any object (`-port`) on the interactive command line shell. For example type:

```
help list_attribute -port
```

Note: Refer to the LeonardoSpectrum Command Reference guide for more information and refer also to the Attributes section in this chapter.

Syntax for Busses

Busses can be specified using the following constraint file syntax:

```
<bus_name>(<start index>:<end index>)
```

Example of bus syntax:

```
arrival_time 10 data(0:3)
```

This example puts an arrival time of `10` on signals `0` to `3` of a bus called `data`.

Alphabetical List

This is an alphabetical list of some attributes/commands allowed in the constraint file. **Note:** Refer also to Command Reference Guide, Attributes chapter, and to HDL Examples section in this chapter.

`arrival_time <time> <signal name>`

Specifies the latest arrival time (nanoseconds) of a signal at an input port.

`buffer_sig <buffer> <signal name>`

Specifies signals to be buffered.

`clock_cycle <clock period> <signal name>`

Specifies the length (nanoseconds) of the clock.

`clock_offset <time> <signal name>`

Specifies the time (nanoseconds) of the leading edge.

`connect <gate name> <instance name> <from pin> <to pin>`

Connects a timing arc inside of a gate or instance for timing analysis and critical path reporting. If the instance name and the gate name contains wildcards, then timing arcs are connected between the *from* pin and *to* pin for all gates with pins that have these names. **Note:** `connect` is removed with the `disconnect` command.

`disconnect <gate name> <instance name> <from pin> <to pin>`

Disconnects a timing arc inside of a gate or instance for timing analysis and critical path reporting. If the instance name and the gate name contain wildcards, then timing arcs are connected between the *from* pin and *to* pin for all gates with pins that have these names.

`dont_touch <instance_name> <true or false>`

Dont touch is used to mark desired instances to prevent unmapping and optimization. In contrast to `noopt`, `dont_touch` prevents optimization of the lower levels of hierarchy and leaf instances. **Note:** Refer to `auto_write` in the Command Reference Utilities chapter.

`input_drive <value> <input signal>`

Specifies the sensitivity to loading of the gate driving an input to the design.

`input_max_fanout <load>`

Specifies the maximum fanout load that the synthesized circuit may present at a design input.

`input_max_load <load>`

Specifies the maximum load that the synthesized circuit may create on an input to the design.

`nobuff <signal name> <true or false>`

Specifies signals that are not buffered internally. Works for input ports only.

`noopt <instance name> <true or false>`

Specifies that an instance should not be optimized or changed. However, in contrast to `dont_touch`, lower level hierarchy and leaf instances are not protected from optimization or change. **Note:** Refer to `auto_write` command in the Command Reference Utilities chapter.

Verilog:

```
//exemplar attribute <module_name> noopt TRUE
```

VHDL:

```
attribute noopt:boolean;

attribute noopt of <component_name>:component is TRUE;
```

`output_fanout <load> <port>`

Specifies the amount of external fanout loads on an output port of the design.

`output_load <load> <port>`

Specifies the number of external unit loads on an output port of the design.

`pad <IO pad type> <signal names>`

Specifies I/O gates to be used for specific signals.

`pin_number <pin number> <port name>`

Assigns a device pin number to a certain port.

`preserve_driver <signal_name>`

Specifies that both a driver and signal name must survive optimization. A driver may be a gate, module, or flip flop instance. The `preserve_driver` attribute tells LeonardoSpectrum to preserve the specific signal and driver in a design. For example, from the input design, any parallel logic, such as a tree of inverters, would be optimized to a single instance. The `preserve_driver` attribute can be applied on these parallel signals which tells LeonardoSpectrum to maintain the parallel structure.

`preserve_signal <signal name>`

Specifies that both a signal and the signal name must survive optimization.

`pulse_width <clock width>`

Specifies the width (nanoseconds) of the clock pulse.

`remove_clock <clock name>`

Removes clocks from the `current_design`.

`required_time <signal> <value>`

Specifies the latest time (nanoseconds) a signal is allowed to arrive at an output port.

`unnoopt <instance name>`

Specifies that the `noopt` symbol is removed from the specified instance.

Attributes

Examples are provided for the following:

- Load and Drive Specifications
- Preserving Signals
- Buffering Specifications
- Timing Requirements
- HDL Examples
- Command Examples

Load and Drive Specifications

Output load and input drive characteristics can be specified for LeonardoSpectrum. All load numbers are in number of unit loads. A unit load is the input load of a single drive inverter in the technology. Thus, the smallest inverter in each technology has an input load of "1". For example:

```
output_load <value> <output signal 1>...<output signal n>
```

The `output_load` command defines the amount of external loading on an output of the design. `<value>` is the number of unit loads driven by the output. This number is used to calculate delays and to ensure that a gate with sufficient drive capability is used to drive an output. If no value is specified for an output, then the default output load for the particular technology is used.

```
max_load <value> <input signal 1>...<input signal n>
```

The `max_load` command defines the maximum load that the synthesized circuit may present at an input to the design. `<value>` is the maximum number of unit loads on the input. If the synthesized circuit exceeds this amount of loading, then a buffer is added to reduce the load.

A buffer or inverter gate with an input load less than the value specified must exist in the target technology, otherwise the program cannot meet the constraint. If no `max_load` value is specified, then no buffering is done for the input signal, unless the technology (for example, Actel) has a global maximum load value. In this case, no load at an input exceeds the specified technology maximum value.

```
input_drive <value> <input signal 1>...<input signal n>
```

The `input_drive` command specifies the additional delay per unit load for an input port. `<value>` is the additional delay in nanoseconds per unit load. This value is used when calculating delays, so that the effects of the load that the synthesized circuit presents to the gate driving the input can be accurately modeled.

Each technology has a default drive defined for inputs, which is usually the drive of a single inverter gate.

```
output_fanout <value> <output signal 1>...<output signal n>
```

The `output_fanout` command defines the amount of external loading on an output of the design. `<value>` is the total number of fanout loads driven by the output.

```
input_max_fanout <value><input signal 1>...<input signal n>
```

The `input_max_fanout` command defines the maximum fanout load that the synthesized circuit may present at an input to the design. `<value>` is the maximum number of total fanout loads allowed. If the synthesized circuit exceeds the number of fanouts, LeonardoSpectrum adds buffers to reduce the load.

Preserving Signals

```
preserve_signal <signal 1> ...  <signal n>
```

The `preserve_signal` command tells LeonardoSpectrum to preserve the specified signal in the Output Design File

If multiple signals implement the same function (for example when a chain of buffers or inverters exist), then any one of those signals may be the one preserved. **Note**: Refer also to `preserve_driver` in the Command Reference guide, Attributes.

The `preserve_signal` command preserves signals, not structure. The gate driving the preserved signal after synthesis may be different from the gate in the original design. To preserve structure, a `noopt` attribute must be added to the specific logic structure in the input netlist.

How to Use the preserve_signal Attribute to Control Timing

The following example illustrates how to control the synthesis process by changing the coding style.

The design consists of a 6-to-64 address decoder and 64 flip-flops. Signal `write_b` enables loading data into one flip-flop at a time. The delay from input pin `write_b` to the generated clocks that drive the flip-flops needs to be minimized.

Note: This example uses gated clocks which may cause glitches; however if the address bus is stable before the write strobe, then the clocks will not have glitches.

Example:

```
package my_pack is
 constant addr_width : integer := 6;
 constant data_width : integer := 2**addr_width;
end my_pack;

library ieee;
use ieee.std_logic_1164.all;
library exemplar;
use exemplar.my_pack.all;
use exemplar.exemplar_1164.all;

entity test is
 port ( addr : in std_logic_vector (addr_width-1 downto 0);
   d_in, write_b, reset_b : in std_logic;
   d_out : out std_logic_vector (data_width-1 downto 0) );
end test;

architecture exemplar of test is
 signal wen_b, dec_out : std_logic_vector
      (data_width-1 downto 0);
 attribute preserve_signal: boolean;
 attribute preserve_signal of dec_out: signal is true;
 attribute max_load: real;
 attribute max_load of write_b: signal is 64.0;

begin
 addr_decode: process (addr)
 begin
     dec_out <= (others => '1');
     dec_out(evec2int(addr)) <= '0';
 end process addr_decode;
 wen_gen: for i in dec_out'range generate
     wen_b(i) <= dec_out(i) or(not write_b);
 end generate;
```

```
--wen_gen: process(addr, write_b)
--begin
 --wen_b <= (others => '1');
 --if (write_b = '0') then
 --wen_b(evec2int(addr)) <= '0';
 --end if;
--end process addr_decode;
```

--But then write_b would be incorporated into the decoder and can take more than --one level of logic from write_b to wen_b.

```
load_data: process (reset_b, wen_b)
begin
 if(reset_b = '0') then
  d_out <= (others => '0');
 else
  for i in d_out'range loop
      if rising_edge(wen_b(i)) then
        d_out(i) <= d_in;
      end if;
  end loop;
 end if;
end process load_data;
end exemplar;
```

Notice the attributes used in this design. Attribute max_load tells LeonardoSpectrum the maximum loading allowed on a signal (write_b). Attribute preserve_signal tells LeonardoSpectrum to preserve the signal and this preserves the implementation.

Alternatively, you can put the attribute clauses in a constraint file and have LeonardoSpectrum read the constraint file. The constraint file default name is <input_filename>.ctr. Write the following to preserve bus dec_out:

```
preserve_signal dec_out(0:63)
```

If the preserve_signal attribute is not used, then LeonardoSpectrum may incorporate write_b in the address decoder. This results in more than one logic level from the primary input signal write_b to the flip-flop clocks.

Buffering Specifications

Signals can be buffered manually with the following constraint file syntax:

```
buffer_sig <buffer_type> <signal_name>
```

In the examples below, the Actel Act2 FPGA device family is the target technology.

```
buffer_sig clkbuf clk1
```

Connects signal `clk1` to the input of the external clock buffer (`clkbuf`), and all the elements which were originally driven by `clk1` will be driven by the clock buffer (`clkbuf`).

```
buffer_sig clkint rstn
```

Connects signal `rstn` (reset signal) to the input of the internal clock buffer (`clkint`), and all the elements which were originally driven by `rstn` will be driven by the clock buffer (`clkint`).

To prohibit the buffering of signals which would otherwise automatically be buffered to meet fanout requirements, the following syntax is used:

```
nobuff <signal1>...<signaln>
```

Timing Requirements

LeonardoSpectrum includes timing analysis routines to decide where to make an area/delay trade off in the logic design.

These routines use your specified timing constraints along with delay information for the library elements, and do a path analysis of the synthesized circuit. Paths start at primary inputs and at register outputs. Paths end at primary outputs and at register inputs. Paths to the asynchronous set and reset of flip-flops are ignored. The latest arrival time and the earliest required time at each node in the network are determined. The difference between these is the slack at the node. A negative slack at a node indicates that the node is on a path which violates some timing constraint.

Input/Output Timing Parameters

You may define the required times at output ports and the arrival times at input ports. The syntax for these commands is:

```
required_time <value> <output port 1> ... <output port_n>
arrival_time <value> <input port 1> ... <input port_n>
```

The required time for an output port defines the longest allowable path from any input port to the output port. Arrival times at primary inputs define the maximum delay to that input through logic external to the synthesized design.

Register Timing

You may define the clock timing for registers (latches and flip-flops). The required and arrival times at the register inputs and outputs are implied through the clock timing definition. The arrival time at a register output is one propagation delay after the leading edge of the clock. The required time at a register input is one setup time before the trailing edge of the clock.

Note: Refer to the Constraint Editor section in this chapter. You can use the editor to set your constraints for: input to register, register to register, register to output, and input to output.

For output ports and register inputs without your specified required times, the required time is set to the value specified with the `-maxdly` option, if specified. Otherwise, the required time is set to the latest arrival time in the circuit. This causes the longest path in the circuit have a zero slack time. The slack times on all other nodes indicate how much faster the worst path through that node is compared with the worst path in the circuit.

You may specify the times in the constraint file and/or VHDL source files. The parameters which may be specified are:

```
clock_offset <value> <clock_signal_1> ... <clock_signal_n>
clock_cycle <value> <clock_signal_1> ... <clock_signal_n>
pulse_width <value> <clock_signal_1> ... <clock_signal_n>
```

These parameters define the behavior of the clock. The behavior of a clock assumes one single common zero reference. `clock_offset` defines the offset of the leading edge from the common zero. `clock_cycle` defines the length of the clock. `pulse_width` defines the length of the clock pulse.

The leading edge of the clock is when new data values appear on the register's outputs. For transparent latches the trailing edge of the clock is when new data is latched in and the values on the input pins must be stable.

For both flip-flops and latches, the leading edge occurs at time `clock_offset`. The arrival time at the register outputs is set to one propagation delay after this time.

For flip-flops, the trailing edge occurs at time `clock_offset + clock_cycle`. The required time at flip-flop inputs is set to this time minus the setup time of the input pin.

Timing analysis assumes that latches are **not** in transparent mode. For latches, the trailing edge occurs at time `clock_offset + pulse_width`. This is the time when the latch goes from being enabled (data passes through) to being disabled (changes on D do not affect Q). The data input is assumed to be stable during the time the latch is enabled. The required time at latch inputs is set to the time of the trailing edge minus the setup time of the input pin.

Clock timing parameters are specified for the actual clock signal (the signal which connects to the clock pin of the register). This signal may be an input port, an output of another register, or the output of some combinational logic. Combinational logic is not a recommended method for generating clocks.

Clock timing is not derived automatically for any signals and must be specified explicitly for each clock. For example, a clock which is a divided down version of another clock must have defined timing specifications. The timing of the source clock does not determine the timing.

HDL Examples

VHDL and Verilog attributes may be used to specify timing parameters in your HDL design.

An example of the syntax of a VHDL attribute is:

```
attribute required_time of out:signal is 10ns;
```

An example of the syntax of a Verilog attribute is:

```
//exemplar attribute intl preserve_signal TRUE
```

The Verilog directive is:

```
//exemplar <object_name> <attribute_name> <value>
```

Command Examples

The following commands are documented in the Command Reference Guide. These commands are used with attributes and constraints.

`list_attributes <object_list>`

List attributes on any object.

`remove_attribute <object_list>`

Remove an attribute from object(s).

`set_attribute`

Create or set an attribute on an object(s).

`report_constraints <design_name>`

Lists user constraints on any object

`set_multicycle_path <-value number> <-from start_point> <-to end_point>`

Constrains a path that requires more than one clock cycle. **Note:** The register-to-register constraint does not work unless you set the clock to make the multicycle paths active.

Low Level Block Constraints

In the bottom up design flow, you set constraints on low level blocks and apply constraints to the boundaries of these blocks. After the design is optimized and read to the top level design, your low level constraints may remain as part of the larger, top level design. These low level constraints are ignored by LeonardoSpectrum unless the low level hierarchy is set to `present_design`, then the constraints are recognized. However, if a black box is part of the low level design, then the black box constraints are used as part of the top level design.

Constraint Editor

The LeonardoSpectrum constraint editor gives you the opportunity to apply and remove constraints to achieve the best design. You can parse the design and setup constraints with the interactive design browser.

- Global constraints (Screen 1 of 8): Specify constraints for the entire design in terms of clock frequency, clock cycle or global path groups.
- Clock constraints (Screen 2 of 8): Specify the clock characteristics of different clocks in the design. Clock specifications include the clock cycle or frequency, pulse width or duty cycle, and clock offset.

Note: You can set a global timing constraint for a required clock frequency in a design. Constraint driven timing optimization attempts to meet the global timing constraint. Clock frequency is determined by paths originating and terminating at registers.

- Input (Screen 3 of 8): Specify the input arrival time and drive characteristics for each input port.
- Output (Screen 4 of 8): Specify the output required time and load characteristics for each output port.
- Signal (Screen 5 of 8): Specify signals to be preserved during optimization.
- Module (Screen 6 of 8): Specify the instance constraints. These are: dont touch, optimize for area or delay, and standard or quick effort.
- Path (Screen 7 of 8): Path based constraints are specified by selecting start points (input ports and registers) and end points (output ports and registers). False paths and multicycle paths can be specified here. **Note:** The register-to-register constraint does not work unless you set the clock to make the multicycle paths active.
- Report (Screen 8 of 8): Generate a report of your constraints, load an existing constraint file, or save current constraints to a file.

Screen 5-1. Constraint Editor - 1of 8 Global

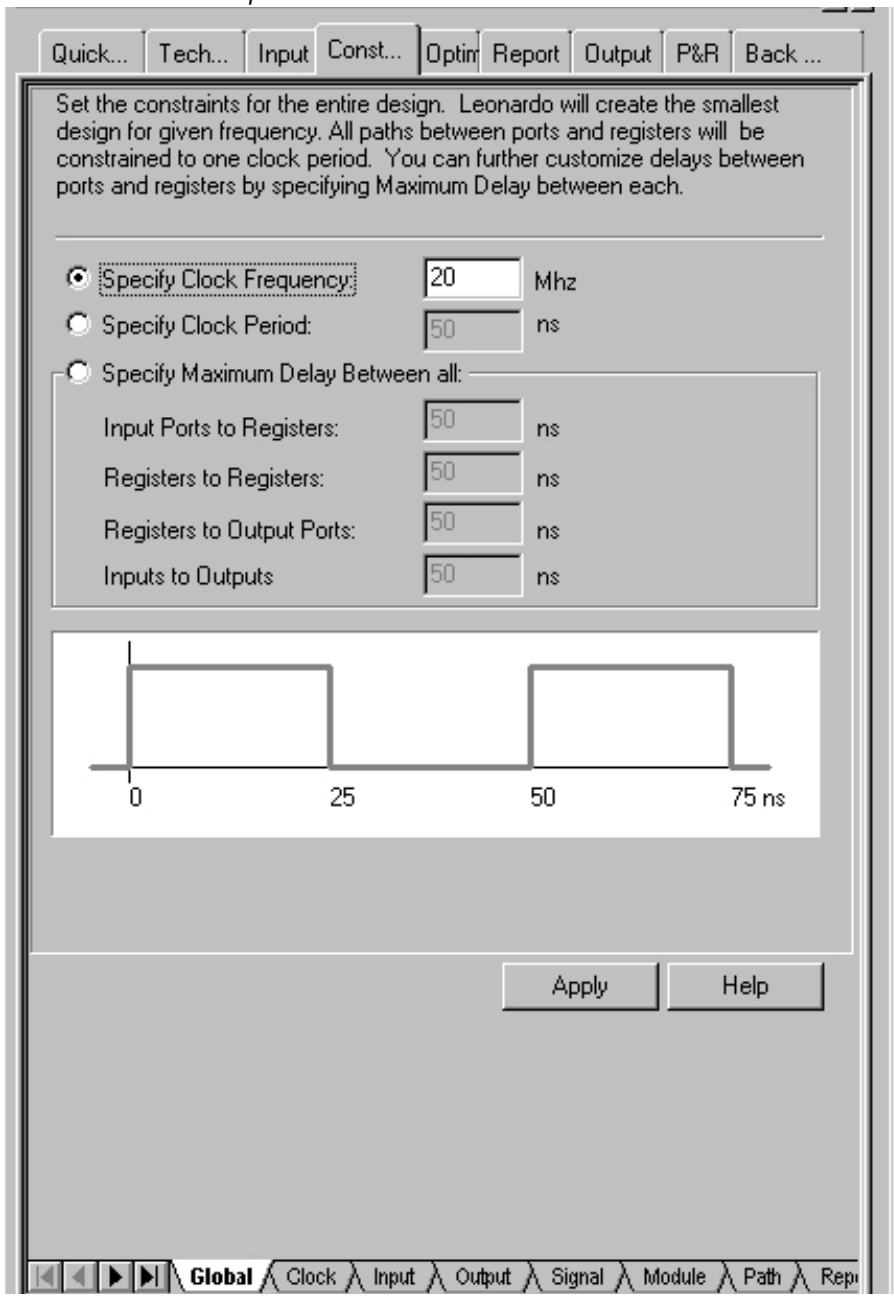
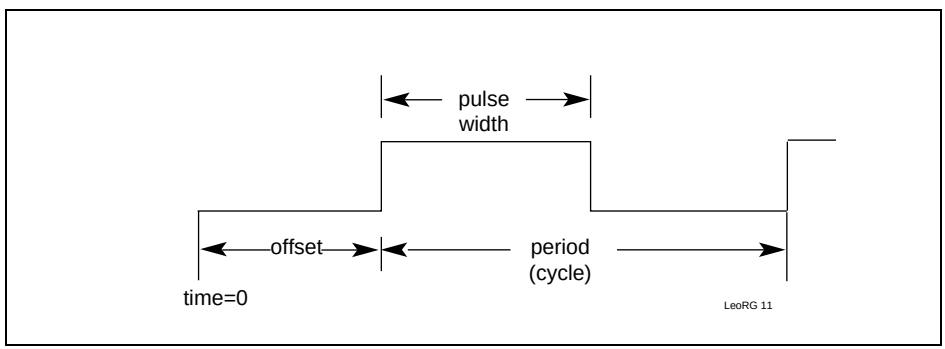


Table 5-1. Global Constraints

Option	Description
●Specify Clock Frequency, (Mhz):	This radio button choice is mutually exclusive with Clock Period. You can specify the required frequency for your design which is 1/period.
❍Specify Clock Period, (ns):	This radio button choice is mutually exclusive with Clock Frequency. See clock period in diagram.
❍Specify Maximum Delay Between all:	This radio button choice gives you control over the delays from port to register, register to register, register to port, and port to port.
Input Ports to Registers:	Delay from input port to input of register in nanoseconds.
Registers to Registers:	Delay from output of one register to input of another register in nanoseconds.
Registers to Output Ports:	Delay from output of register to output port in nanoseconds.
Inputs to Outputs	Delay from input port(s) to output port(s).
Waveform Window	The clock pulses show your settings.

Click **Apply** to apply settings to your design. Click **Help** to open online help.



Screen 5-2.Constraint Editor - 2 of 8, Clock

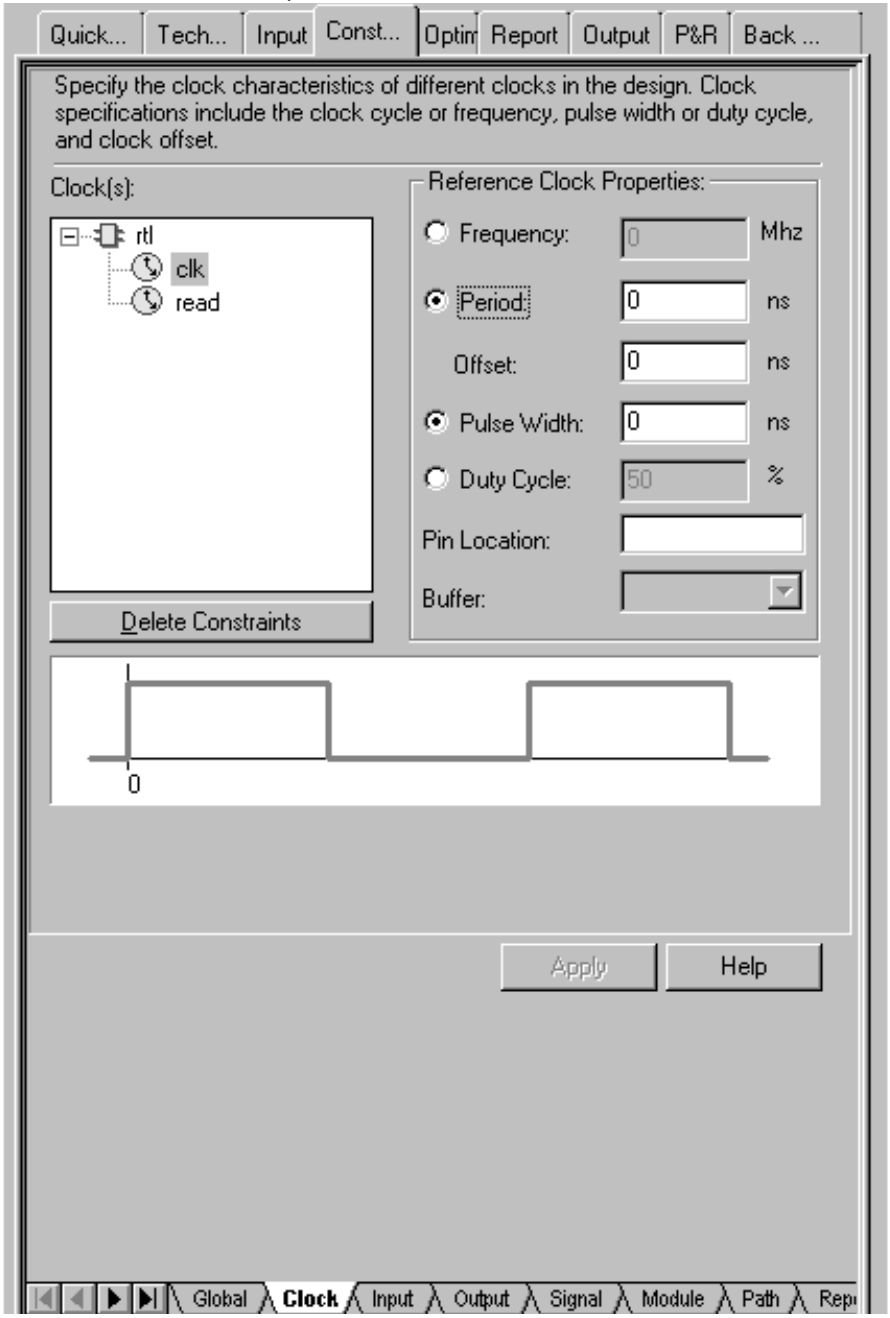
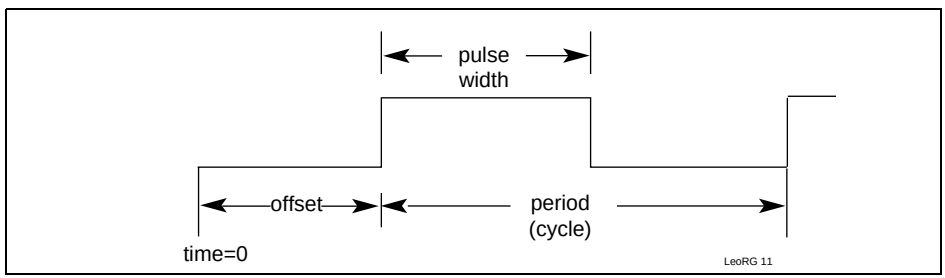


Table 5-2. Clock Constraints

Option	Description
Reference Clock Properties:	The Reference Clock is **relative** to the signal. Every signal is measured relative to the reference clock. Reference Clocks are saved in your constraint file.
●Frequency: (2 digits of accuracy)	Frequency (Mhz) is mutually exclusive with period.
OPeriod: (2 digits of accuracy)	Period is a waveform that repeats at fixed intervals (ns).
Offset, ns: (Offset of Leading Edge)	This is the delay after time=0. Timing is **absolute**. Timing is offset in nanoseconds from time=0.
●Pulse Width, ns: (Duration of Pulse)	Pulse width is a measure of the duration of the pulse in nanoseconds.
❍Duty Cycle, %: (Duration of Pulse)	Duty cycle percentage is equal to the pulse width divided by the period times 100. Some pulses do not repeat at fixed intervals. The pulse widths and time intervals may differ.
Pin Location:	This is the equivalent of the PIN_NUMBER attribute.
Buffer: (BUFG, None)	Select None to imply that ports are not assigned pads. BUFG I/O pads are available for selected technology. Equivalent of the BUFFER_SIG attribute.
Click **Apply** to apply settings to your design. Click **Help** to open online help.	
Clock(s)	Browse through the interactive, filtered list of clocks. This list was built when you read in your design.
Delete Constraints Button Click **Apply** to apply deletions.	Select an object and click **Delete** and then **Apply.** All constraints set on this object are deleted.
Waveform Window	This is a display of your clock settings.



Screen 5-3. Constraint Editor - 3 of 8 Input

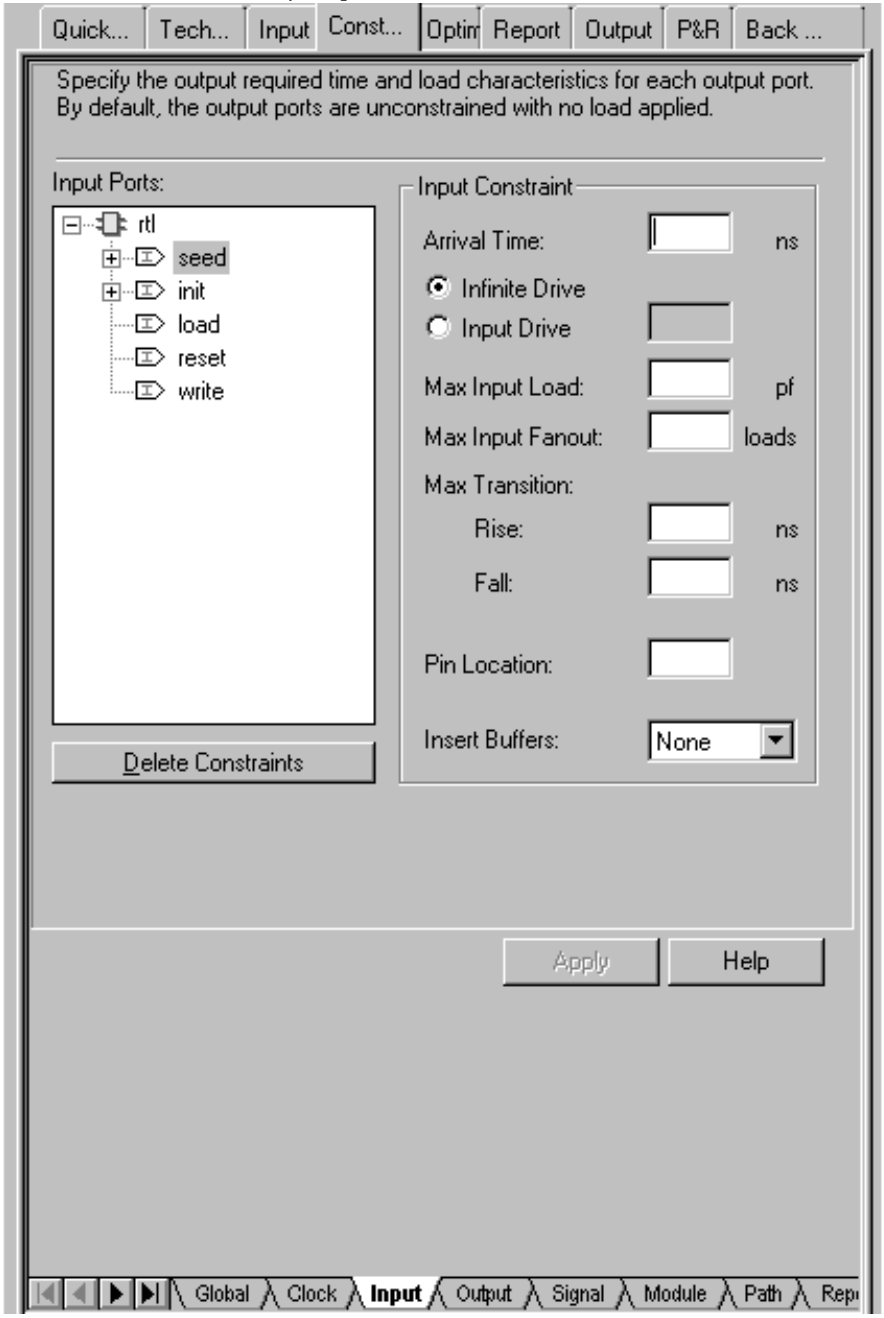


Table 5-3. Input Constraints

Option	Description
Input Constraint	Specify the input arrival time and drive characteristics for each input port. The default is 0 ns arrival time and infinite drive.
Arrival Time: (ns)	Select a signal for your reference clock. Arrival times at primary inputs define the maximum delay through logic external to the design before arrival at that input.
●Infinite Drive	This is the default when arrival is 0 ns and load is 0.
❍Input Drive	Select this option to specify the additional delay per unit load (ns) for the selected input port(s). This option allows an accurate modeling of the effects of the load presented at the gate by the synthesized circuit.
Max Input Load: pf (pico farad)	This is the capacitance load for your gates. The load input controls operation of the output. If the synthesized circuit exceeds the number of loads, then LeonardoSpectrum buffers the load. A buffer or inverter gate with an input load less than the value specified must exist in the target technology to meet the constraint. If the technology has a global maximum load value, then an input cannot present a load at an input that exceeds the technology maximum.
Max Input Fanout: (loads)	Fanout is the number of loads that the output of a gate can drive. If the synthesized circuit exceeds the number of loads, then LeonardoSpectrum buffers the load.
Max Transition: ns (Rise)	This is the rise time in the leading edge of the pulse. This is the time required for the pulse to go from low level to high level.
Max Transition: ns (Fall)	This is the fall time in the trailing edge of the pulse. This is the time required for the pulse to go from high level to low level.
Pin Location:	This is the attribute PIN_NUMBER.
Insert Buffers: (Global, none, SCLK)	I/O Pads available for selected technology. This is the attribute BUFFER_SIG.
Delete Constraints Button: Click **Apply** to apply deletions.	Select an object and click **Delete** and then **Apply.** All constraints set on this object are deleted.
Input Port(s): window	Browse through the interactive, filtered input port list. This list was built when you read in your design.

Click **Apply** to apply settings to your design. Click **Help** to open online help.

Screen 5-4. Constraint Editor - 4 of 8 Output

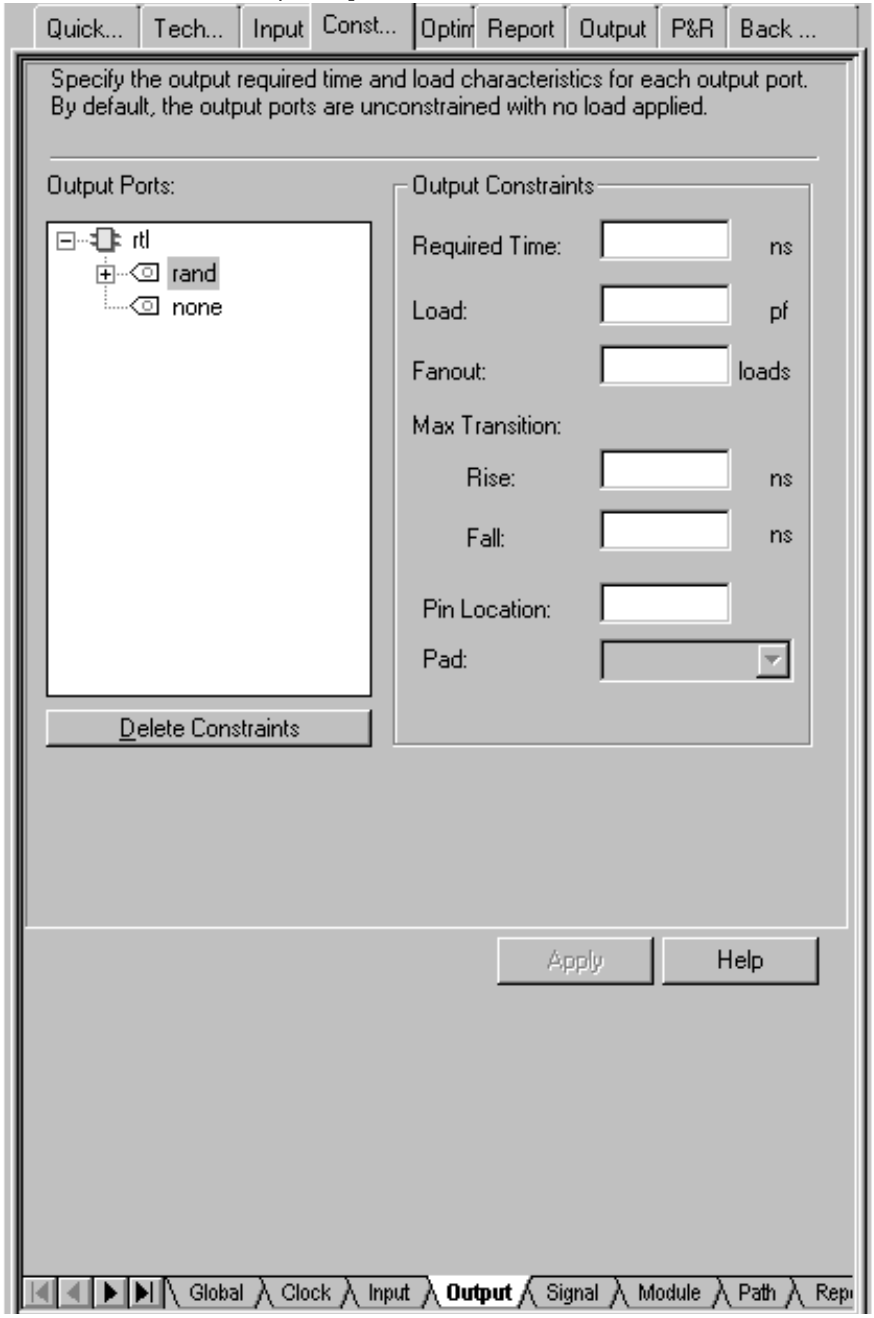


Table 5-4. Output Constraints

Option	Description
Output Constraints	Specify the output required time and load characteristics for each port. The default is output ports with no load applied.
Required Time (ns):	These choices allow you to constrain the output ports and apply the required loads. The output clock is relative to the output port. This is the time required for a signal to be available at this port.
Load: (pf), pico farad: (Number of loads driven by output)	This is the capacitance load of the output port. Specify the amount of external loading on the design output. The loads value is used to calculate delays and to ensure that sufficient drive capability is available at an output. Meeting load and drive requirements may require choosing a gate with higher drive or replicating logic. The default output load for the technology is used, if a load value is not specified.
Fanout: loads: (Number of loads driven by output)	The output fanout depends on the input. Fanout specifies the maximum loading a gate can handle. Specify the number of external fanout loads driven by the output.
Max Transition: ns (Rise)	This is the rise time in the leading edge of the pulse. This is the time required for the pulse to go from low level to high level.
Max Transition: ns (Fall)	This is the fall time in the trailing edge of the pulse. This is the time required for the pulse to go from high level to low level.
Pin Location:	This is the attribute PIN_NUMBER.
Pad: (pull down: 0BUF, OBUFT, None)	Select None to imply that ports are not assigned pads. BUFG I/O pads are available for selected technology. Equivalent to the PAD attribute.
Delete Constraints Button: Click **Apply** to apply deletions.	Select an object and click **Delete** and then **Apply.** All constraints set on this object are deleted.
Output Port(s): window Interactive filtered list	This list of ports is available in your design. Double click on a port name to select. This list was built when you read in the design.

Click **Apply** to apply settings to your design. Click **Help** to open online help.

Screen 5-5. Signal - 5 of 8 - Signal

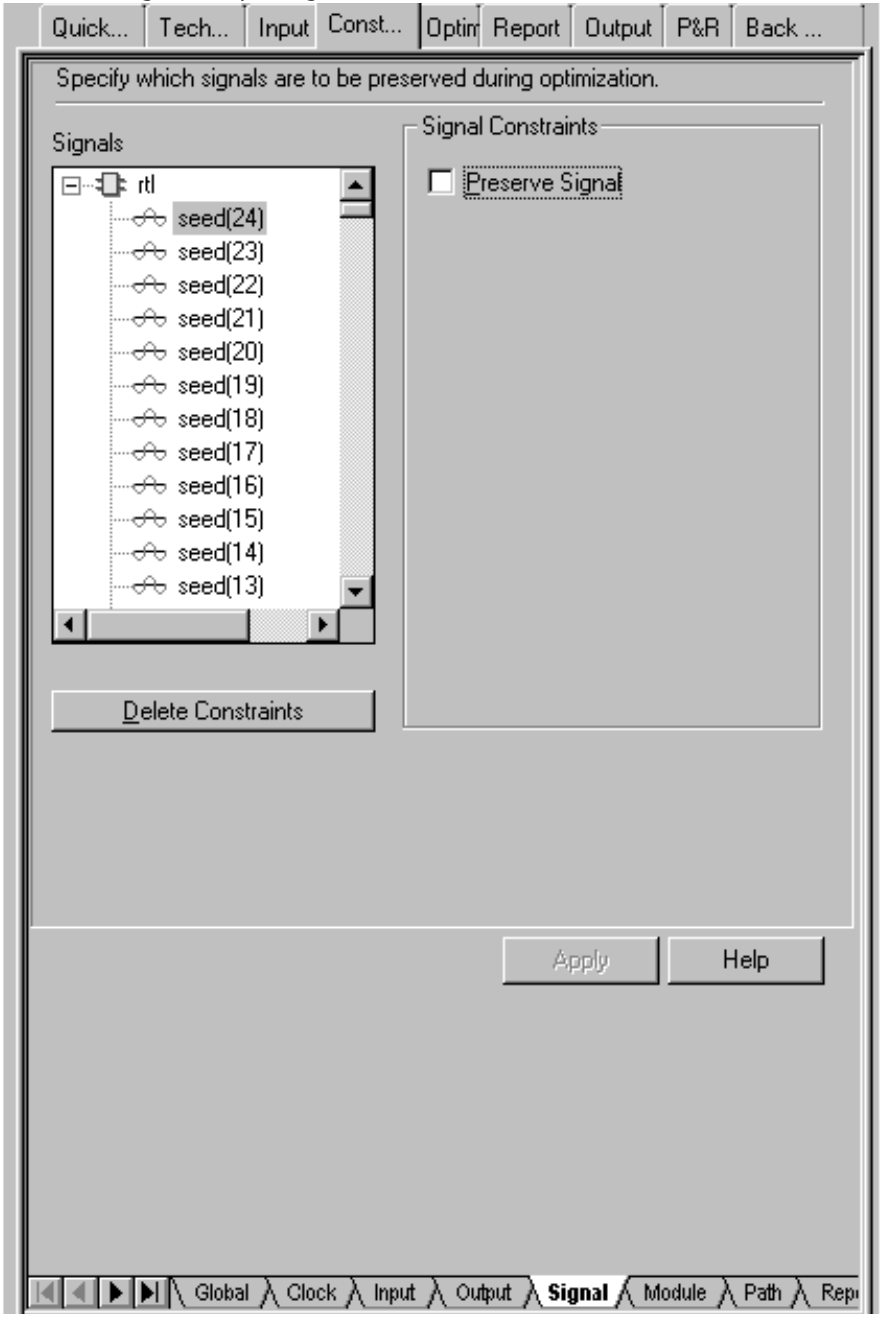


Table 5-5. Signal

Option	Description
Instance Constraints	Specify which signal to preserve during optimization.
❑Preserve Signal	If selected, your customized signal is preserved during optimization.
Signals Scroll Window	This is an interactive, filtered design browser list of signals. This list was built when the design was read.
Delete Constraints Button Click **Apply** to apply deletions.	Select an object and click **Delete** and **Apply.** All constraints set on this object are deleted.
Click **Apply** to apply settings to your design. Click **Help** to open online help.	

Screen 5-6. Module - 6 of 8 Output

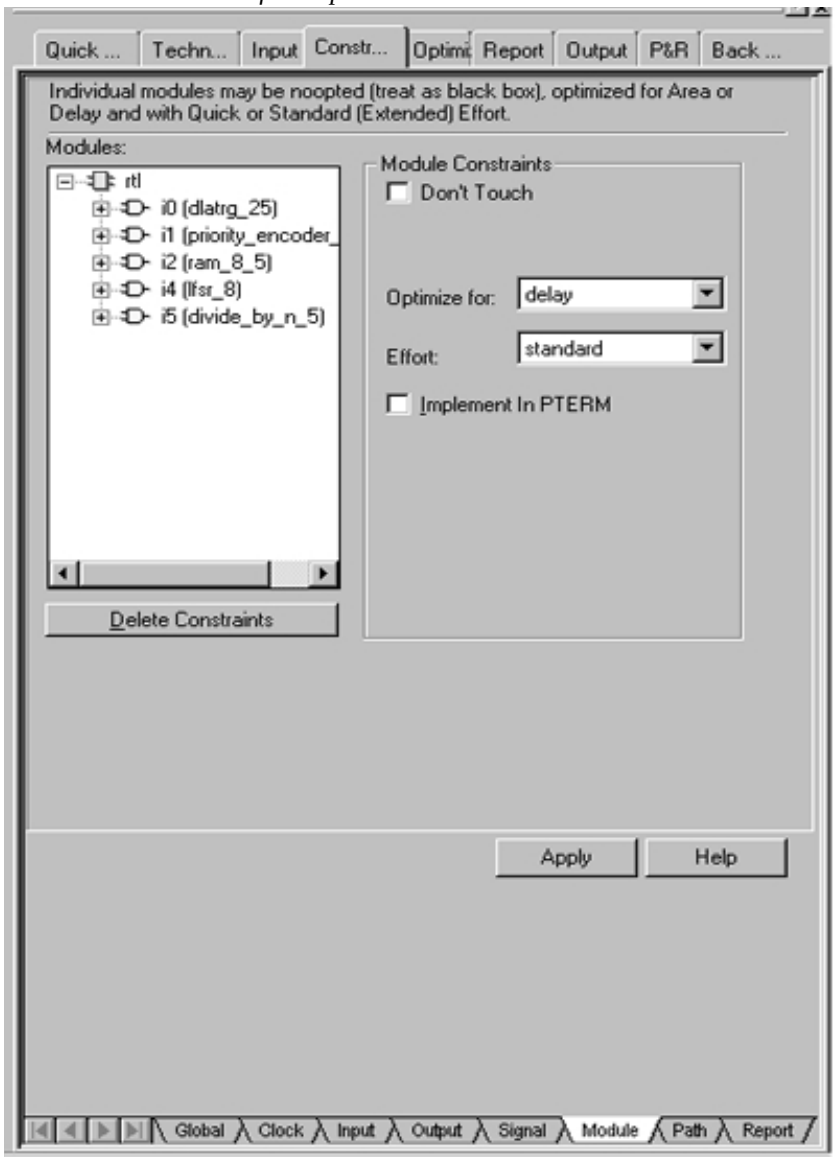


Table 5-6. Module - 6 of 8 Output

Option	Description
Module Constraints	
☐Don't Touch (Refer to auto_write in Command Reference Utilities chapter.)	If selected, the specified technology cells are not optimized (NOOPT). dont_touch is an attribute used for preserving custom implementations and technology instantiation; used for optimizing hierarchically and for protecting buffering.
Optimize for: (pull down with area, delay)	Choose from pulldown. Choose area. The circuit is optimized to minimize area and not delay; or choose delay to optimize for speed and not area. The default is area.
Effort: (pull down with quick, standard)	Choose from pulldown. Choose quick. Only one optimization strategy is attempted on the network; or choose standard to run multiple optimization algorithms. The default is quick.
☐Implement In PTERM	When ■Implement In PTERM is selected, then the block to be mapped to PTERM is flattened by default.
Modules:	This is a filtered, interactive list from the design browser of the modgen library instances. This list was built when you read in the design.
Delete Constraints Button Click **Apply** to apply deletions.	Select an object and click **Delete** and **Apply.** All constraints set on this object are deleted.
Click **Apply** to apply settings to your design. Click **Help** to open online help.	

Screen 5-7. Constraint Editor - 7 of 8 - Path

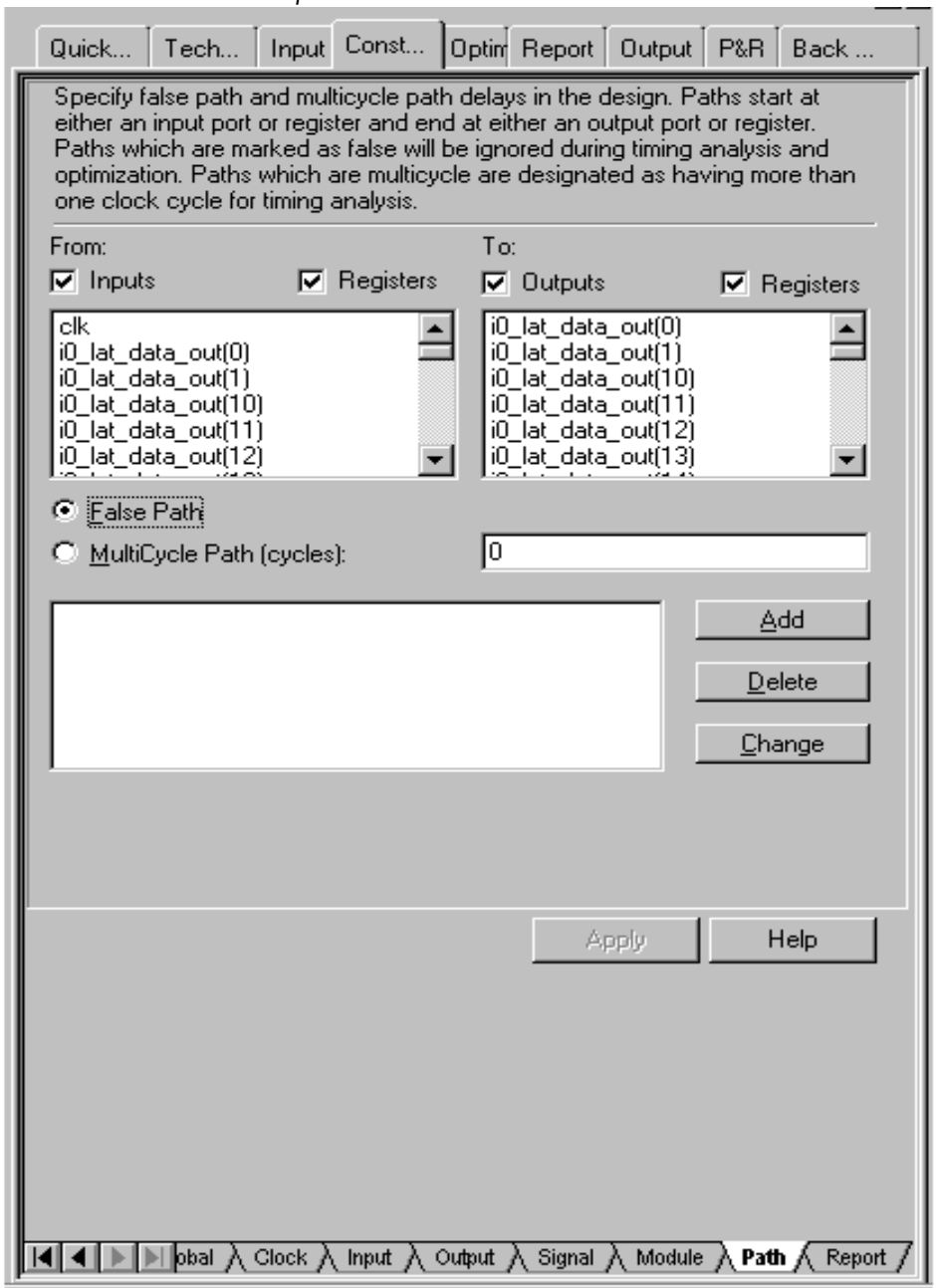


Table 5-7. Path Constraints

Option	Description
From:	This is the starting point for a multicycle or absolute path. A list of filtered inputs and registers is displayed if the check boxes are selected. Double click to select.
■ Inputs	The input ports in your design are shown in the list of objects. Click to select these ports.
■ Registers	The registers in your design are shown in the list of objects. Click to select the starting input port(s) and/or register(s) on which to apply path constraints from the list of objects. Paths in your design may be assigned various constraints.
To:	This is the end point for a multicycle or absolute path. A list of outputs and registers is displayed if the check boxes are selected. Double click to select.
■ Outputs	The output ports in your design are shown in the list of objects. Click to select these ports.
■ Registers	The registers in your design are shown in the list of objects. Click to select the starting output port(s) and/or register(s) on which to apply path constraints from the list of objects. Paths in your design may be assigned various constraints.
●False Path:	False path disables timing in nanoseconds between points. Rising or falling transition, and setup or hold can be disabled. False paths are ignored during timing analysis.
❍MultiCycle Path (cycles):	Constrains a path that requires more than one clock cycle. You can specify multiples of cycles in the field. Multicycle paths have more than one clock cycle for timing analysis. Specify the number of clock cycles.

Click **Add** to add an input to output path. Highlight an existing path, click **Delete** to delete the path. Highlight an existing path, complete your From: To: changes, and click **Change** to change a path. Click **Apply** after **Add**, **Delete**, **Change**.

Summary Window Example: `lat_mem[1][0]...lat_mem[0][3][2]`

Click **Apply** to apply settings to your design. Click **Help** to open online help.

Screen 5-8. Constraint Editor - 8 of 8 - Report

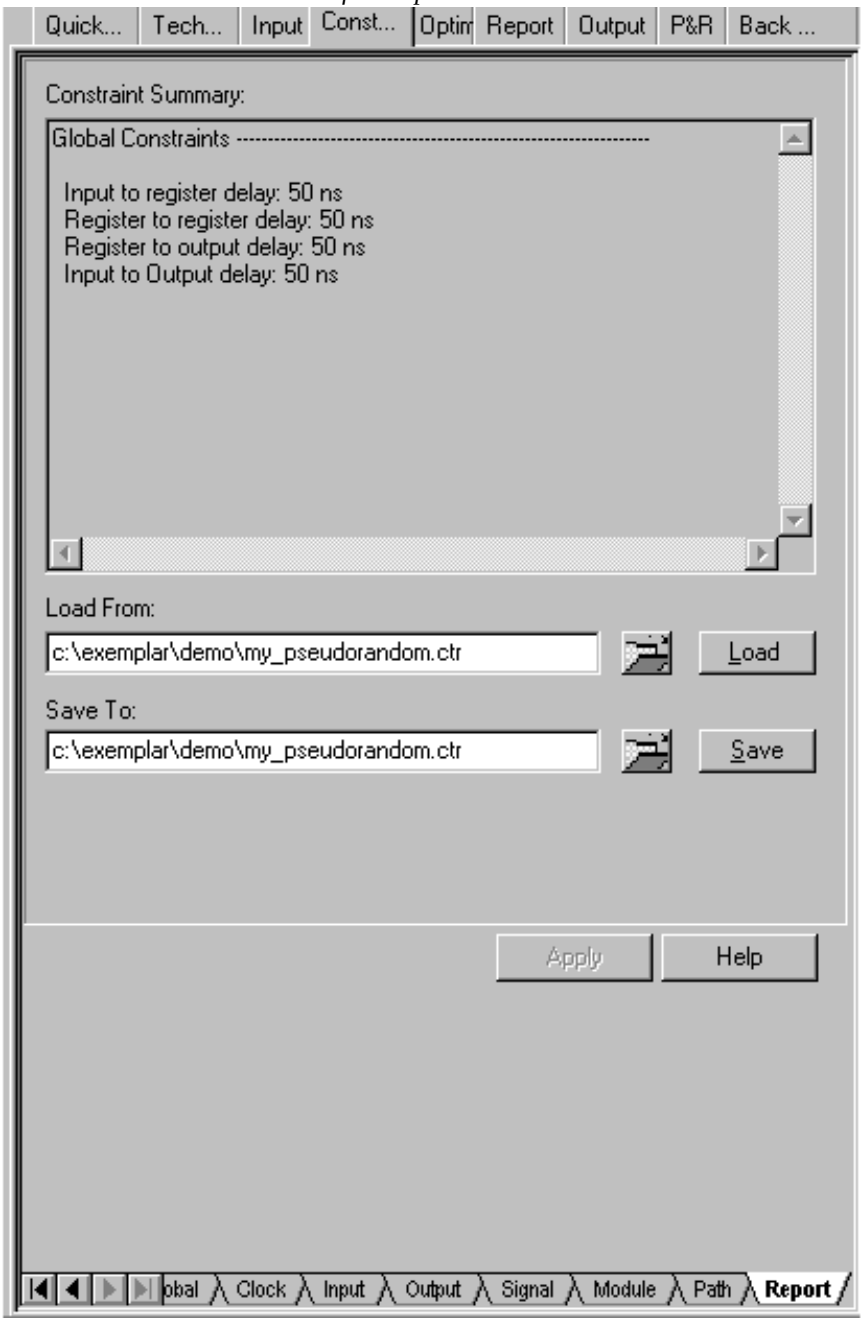


Table 5-8. Constraints Report

Option	Description
This report shows all the constraints set on your design. You can save these constraints to a file or load a file of previously saved constraints.	
Load From:	Click on folder 🗁to bring up Loading Constraint File. Select or enter constraint file from Files of type: with *.ctr extension. Use the Windows browser, click, and drag rules as required. Click on Load button.
Save To:	Click on folder 🗁to bring up Saving Constraint File. Select or enter constraint file with *.ctr extension. Use the Windows browser, click, and drag rules as required. Click on Save button.
Click **Apply** to apply settings to your design. Click **Help** brings up online help.	

Special Steps for Quick Setup - Level 2

When the working directory contains a `<design_name>.ctr` file, then this file is read in automatically. For example, you can set up a pseudorandom.ctr file on Report tab. Use these steps to apply a constraint file to your top module or entity on the Quick Setup tab:

1. On the Technology tab, load the library for your selected technology. Load the source file on the Input tab. Select constraints. Run the Optimize tab, and review your accumulated constraints on the Constraints Report tab.
2. Save the `<design_name>.ctr` file with the SAME name as the top module or entity file name. For example, if you are running the flow for pseudorandom.vhd (top entity) demo, then name your constraint file `pseudorandom.ctr`.
3. On the Quick Setup tab, check your settings for the `pseudorandom.vhd` and then run the flow. LeonardoSpectrum applies your constraints file. Check the Information window. The constraints in the `pseudorandom.ctr` file have been applied.
4. You can also run the entire flow in batch mode and apply your constraints by using the `-control=<string>` batch mode command.

Note: The constraints file, `pseudorandom.ctr`, must have the same name top entity or top module name as pseudorandom.vhd input file. The extension must be *.ctr.

Batch Mode Options 6

This chapter presents the options and switches for LeonardoSpectrum batch mode. **Note:** Refer to the Command Reference guide for more information. This chapter is divided as follows:

- Alphabetical List
- Options for MAX+PLUS II Functionality
- License Information
- Tcl Script Sourcing

Alphabetical List

This is an alphabetical list of options and switches. Each entry in the list is followed by a brief explanation. The defaults in the list are identified. The command line syntax is:

```
spectrum <input file> <output file> <-target technology> [more options]
spectrum -file <my_script_file> (Run your Tcl script file in batch mode)
spectrum -product ls2|ls1 -file <my_script_file>
```

-architecture=<string>

This option, when used with the `-entity=<string>` option, defines the top level of hierarchy in the input VHDL design. This is a VHDL only option. By default, LeonardoSpectrum picks up the last architecture in the design to be synthesized if this option is not specified. The `<string>` is case sensitive and must be specified in lower case only. In VHDL, a mixed case or all upper case architecture name cannot be used.

`-area | -delay`

Directs LeonardoSpectrum to optimize the circuit to minimize area rather than delay. The `-area` option is mutually exclusive with the `-delay` option. `-area` is the default when neither `-area` nor `-delay` is specified. The `-delay` option directs LeonardoSpectrum to optimize the circuit to minimize delay rather than area.

`-batchhelp`

Type this option to display a list of all batch mode options. Type:
`$EXEMPLAR\leospec\spectrum -batchhelp`

`-bus_name_style=<string>`

This option allows you to customize bus names in the EDIF output. This is the naming style for vector ports and nets. For example: default %s(%d) or a(0), simple %s%d or a0, or a_0. Example:
`-bus_name_style %s%d|simple %s%d|old_galileo %s_%d`

`-chip | -macro`

The `-chip` option directs LeonardoSpectrum to add I/O buffers or preserve I/O buffers around the periphery of the design. The `-chip` option is mutually exclusive with the `-macro` option. `-chip` is the default when neither `-chip` nor `-macro` is specified.

The `-macro` option specifies that the input design represents a part of a complete design, for example a user level of hierarchy. When the macro mode is specified, I/O buffers are not added to the design. The `-macro` option is mutually exclusive with the `-chip` option.

`-command_file=<list> | [{<file_name>}]...[{<file_name>}]`

Specifies options from a separate file, instead of from the command line. Using this option causes LeonardoSpectrum to read additional command line options from this file. This option can be used multiple times or can accept a list of files as a parameter.

-control=<string>
-nocontrol

Specifies design specific constraints to LeonardoSpectrum. For example, the arrival time at the inputs, the required times at the outputs, the load at the output ports, and the pads which should be connected to a signal are a few of the design constraints that may be specified in a control file. By default, LeonardoSpectrum looks for <input_filename>.ctr as the control file. This option can be used to specify a different file, or use the -nocontrol option to override the use of the default control file.

-crit_path_analysis_mode=<string> (maximum|minimum|both)

Directs LeonardoSpectrum to analyze and report setup violations (maximum) or hold violations (minimum) or both. The default is maximum.

In the maximum delay analysis mode, worst case (maximum) arrival times and delays through all the gates are used, and timing violations at the outputs and setup violations at the register inputs are reported.

In the minimum delay analysis mode, best case (minimum) arrival times and delays through all the gates are used. Hold violations at the register inputs are reported.

-crit_path_arrival=<float>

Specifies a threshold for the arrival time (ns). Only paths with arrival times greater than this number are reported.

-crit_path_detail=short|full

Controls the level of detail in the critical path report. A critical path report with full detail gives a point-to-point report of the entire path. A short report gives only the start point and the end point of a path. The default is full.

-crit_path_from=<list>
-crit_path_to=<list>

These are filters which direct LeonardoSpectrum to report critical paths starting or ending at specific points (instances, nets, ports). Any number of start or end points can be specified by providing a list of start or end points as parameters to these options or

by repeating these options. When `-crit_path_from` is used, only critical paths starting at these start points are reported. When `-crit_path_to` is used, only critical paths ending at these points are reported.

`-crit_path_longest`

This option directs LeonardoSpectrum to show the longest path first rather than the most critical path. The paths are sorted by arrival time, with latest arrival time first, rather than by slack.

`-crit_path_no_int_terminals`

This option filters out paths that terminate internally (paths that terminate at register inputs or black boxes) and reports paths terminating in primary outputs only.

`-crit_path_no_io_terminals`

This option filters out paths that terminate in primary outputs. Only paths that terminate at inputs of registers or black boxes are reported.

`-crit_path_report_input_pins`

Reports input pins of gates in the critical path report. By default, this option is turned off, and only output pins are reported.

`-crit_path_report_nets`

Reports nets in the critical path report. By default, this option is turned off. When this option is turned on, the number of fanouts of the net are also reported in the last column, instead of the load.

`-crit_path_rpt=<string>`
`-nocrit_path_rpt`

Specifies the critical path report file. By default, this is `<output_filename>.rpt`. The `-nocrit_path_rpt` option prevents the critical path report from being created.

`-crit_path_slack=<float>`

Specifies the slack threshold. Paths with slack less than the slack threshold are considered critical. The default slack threshold is 0.0, so all paths with negative slack are critical by default.

`-crit_paths_thru=<list>`
`-crit_paths_not_thru=<list>`

`-crit_paths_thru` is a filter that reports critical paths through a particular instance, net, port, or port instance (pin). Any number of points can be specified by repeating this option or by specifying a list of points. When points are specified with this option, only critical paths that pass through these points are reported. `-crit_paths_not_thru` is a filter that reports only critical paths that do not pass through the specified points.

`-design=<string>`

When reading an EDIF netlist, LeonardoSpectrum assumes by default that the root (top level) cell of the design has the same name as the input file. This option allows a different root cell name to be specified.

`-dont_lock_lcells | -lock_lcells`

This option either locks or does not lock LCells for Altera and Xilinx CPLD technologies. The default is `-lock_lcells` for Altera and `-dont_lock_lcells` for Xilinx CPLD. The default dont_`lock_lcells` allows LeonardoSpectrum to set KEEP attributes (in EDIF) or attach OPT_OFF symbol (in XNF) on the output of the macrocell. The default `lock_lcells` directs LeonardoSpectrum to avoid putting certain attributes in the EDIF file that locks LCELLS in MAX+PLUS II.

`-edif_file=<string>`

An EDIF netlist can be written out as a second output netlist with this option. This is useful when a VHDL or Verilog simulation netlist is produced and an EDIF netlist is also needed for schematic viewing.

`-edif_adl_flavor`

Write out EDIF with legal adl naming.

`-edif_timing_file=<string>`

Write out EDIF file for timing analysis.

`-edifin_ground_net_names=<list>`

Specify that net(s) with given name(s) are ground nets.

`-edifin_ground_port_names=<list>`

Specify that port(s) with given name(s) are ground ports.

`-edifin_ignore_port_names=<list>`

Specify that port(s) with given name(s) ignore ports.

`-edifin_power_net_names=<list>`

Specify that net(s) with given name(s) are power nets.

`-edifin_power_port_names=<list>`

Specify that port(s) with given name(s) are power ports.

`-edifout_ground_net_name=<string>`

Special name for ground nets when `-edifout_power_ground_style_is_net` is `true`, default `GND`.

`-edifout_power_ground_style_is_net`

Write out power and ground as undriven nets with special names.

`-edifout_power_net_name=<string>`

Special name for power nets when `-edifout_power_ground_style_is_net` is `true`, default `VCC`.

```
-effort=<string>
[-reformat]|[-remap]|[quick]|[-standard]|[-effort <string>]
```

Optimization effort:

```
analyze|back_annotate|reformat|remap|quick|standard
```

This switch controls the level of effort applied to optimizing the design. Choices for `effort=<string>` are:

`quick`

Attempts only one optimization strategy on the network. This is much faster than running with the standard option, but may not produce as good a final result. This is the default if no effort option is specified.

`reformat`

Instructs LeonardoSpectrum to reformat the design from the source to the target netlist format. This option does not do an optimization. Can only be used when the source and target technologies are the same. May also be used to determine the size of a design before optimization by LeonardoSpectrum.

`remap`

Does not attempt to optimize the network, but simply maps it into the target technology. This is useful when the input design is already optimized and mapped to some technology and the design needs to be mapped into a new technology. This option usually results in inferior designs when the input format is technology independent, such as VHDL or Verilog.

`standard`

Runs multiple optimization passes on the design. This is slower than running with the quick option, but may produce better results since it explores more of the design space.

```
analyze (optimization effort)
```

```
back_annotate(optimization effort)
```

```
-enable_dff_map_optimize
-noenable_dff_map
```

Enable clock enables from random logic, while noenable disables inferences of clock enable DFFs.

`-encoding=<string>`

This switch controls the style of state machine encoding applied to a VHDL or Verilog design. When there are N states, `onehot` encoding results in N state registers. All other encoding strategies result in log2(N) state registers. **Note:** Verilog designs use the “enum” attribute. The choices for `<encoding_style>` are:

`binary`

Encoding is done based on the definition of the state type, counting left to right. In this example, the following state values are assigned to each state:

state	state-bit 012
s0	000
s1	001
s2	010
s3	011
s4	100

Change the order of the enumeration values to achieve different binary encoding values for each of the states.

`gray`

Adjacent enumeration values differ only by one bit.

`random`

Values are encoded in random order (reproducible).

`onehot`

Each state is assigned a state register. The encoding is one-bit-per-value. Only a single bit is '1' at any given time.

`twohot`

Two flip flops are set high for each state. This is for large FSMs.

`auto`

For some technologies (Altera FLEX 6/8/10 and Xilinx 4000/5000), LeonardoSpectrum varies the encoding based on bit width. More specifically, enumerated types with fewer elements than global integer lower_enum_break are encoded as binary, larger enumerated types are encoded as onehot. Values

larger than global integer upper_enum_break are encoded as binary. On the GUI, `auto` may be the default. `auto` allows LeonardoSpectrum to select encoding on a case by case basis.

`-entity=<string>`

Defines the top level of hierarchy in the VHDL design. See also `-architecture=<name>`.

`-exclude=<list> | [{<gate_name>}]...[{<gate_name>}]`

Directs LeonardoSpectrum not to use the gate `<gate_name>` when mapping the design to the target technology. `<gate_name>` must be a gate in the target technology. This option can be used multiple times, or can accept a list of gates as the parameter. **Not available for:** Xilinx, Altera FLEX and MAX, Lucent ORCA; and in some cases, not available for Actel technologies. **Note:** If modgen instantiates a gate, exclude will not filter it.

`-file=<script_name>`

Runs a specified script. For example name your script `try.tcl` and type: `spectrum -file try.tcl.`

`-full_case`

If a case statement is used in the input Verilog, this option specifies to LeonardoSpectrum that all conditions of the case statement are specified. If no default assignment was used, then this option prevents the implementation of extraneous latches. **Note:** Refer also to `-parallel_case`

```
-generic=<list> <value> |
[{-generic=<name> <value>}]...[{-generic=<name> <value>}]
```

When using VHDL as input to LeonardoSpectrum, this option allows the designer to set the value for the specified generic(s). This option can be used multiple times or can accept a list of generics as a parameter.

`-global_sr=<string>`

Specify a signal name as global set/reset. Applies only to ORCA 2CA/3C/2TA and Xilinx 4000/E/EX/XL and 5200 target technologies. This option disables automatic inference of global set/reset. **Note:** Must be active high reset signal.

`-help`

Displays a list of the command line options. You can also type `-batchhelp`.

`[-hierarchy_flatten]|[hierarchy_preserve]|[hierarchy_auto]`

This option flattens, preserves or auto dissolves your design hierarchy during optimization. Default is `hierarchy_auto` (auto dissolve).

`-highlight_file=<string>`

Critical path highlighting file for Netscope.

`-include=<list> | [{<gate_name>}]...[{<gate_name>}]`

Applies only to ORCA. Do not exclude gates that are predefined in the target technology. This option can be used multiple times or can accept a list of gates as a parameter.

`-insert_global_bufs`

Use global buffers for clocks and other global signals for Xilinx and Actel.

`-input_format=<string>`

Specifies the format of the input design. This is optional if the input format can be determined from the extension of the input filename. Supported input file types and corresponding extensions are:

file_type	extension	description
XNF	.xnf	Xilinx Netlist Format
EDIF	.edf, .edif, .eds, .edn	EDIF netlist
VHDL	.vhd, .vhdl	VHDL
Verilog	.v, .vg, .vlg, .verilog	Verilog HDL

Refer also to the `-output_format` option.

Both of the following specifications read a VHDL file:

```
input_filename1.foo -input_format=VHDL
input_filename2.vhd
```

`-logfile=<string>`
`-nologfile`

Give the logfile a name or do not generate a logfile.

`-lut_max_fanout=<integer>`

Specify the fanout of the net for LUT technologies, Xilinx, Altera FLEX, ORCA. Refer also to the Command Reference guide.

`[-map_area_weight <float>]|[-map_delay_weight <float>]`

Specifies an integer between 0 and 1.0. The greater the number, the more mapping tries to minimize area. This allows finer control over the `-area` or `-delay` switch.

`-map_muxf5`

Map to MUXF5 for Xilinx XCV technology.

`-map_muxf6`

Map to MUXF6 for Xilinx XCV technology. Implies mapping to MUXF5.

`-maxarea=<float>`

Specifies the maximum area acceptable for the optimized circuit. This directs LeonardoSpectrum to search for the fastest circuit implementation which meets the specified area constraint. LeonardoSpectrum tries to find a solution that meets this constraint; however, finding a solution cannot be guaranteed.

`-max_cap_load=<float>`
`-max_fanout_load=<float>`

Use `-max_cap_load` to override the default `capacitive_load` specified in the library. Use `-max_fanout_load` to override the default `fanout_load` specified in the library. Applies to Actel and QuickLogic technology libraries only.

`-maxdly=<float>`

Specifies the maximum delay acceptable for the optimized circuit. This directs LeonardoSpectrum to search for the smallest circuit implementation that meets the specified timing constraint. LeonardoSpectrum attempts to find a solution that meets this constraint; however, finding a solution cannot be guaranteed.

`-max_fanin=<integer>`

This option controls the maximum fanin into a function block when targeting lookup table FPGA technologies like Xilinx, Lucent and Altera.

`-max_frequency=<float>`

This option sets the maximum clock frequency timing constraint in Mhz, for all global clocks in the design. Integer is 1 to 9999.

`-max_pt=<integer>`

This option controls the maximum number of product terms in a function. Applies to Altera MAX and Xilinx 9500 target technologies only.

`-modgen_library=<list>`

This option overrides the use of the technology-specific modgen library for the target technology. A different external modgen library (for example, XBLOX3 or XBLOX4) can be specified, or `-modgen_library=none` can be used to prohibit the use of any external modgen library. If none is specified, or if there is no technology specific library for the target technology, then default internal module generation routines are used.

`-module=<string>`

This option allows the user to specify the top-level module in the hierarchy of a Verilog HDL design.

`-ncf_filename`

Generate the netlist constraint file for Xilinx.

`-nobreak_loops_in_delay`

Directs LeonardoSpectrum NOT to break combinational loops statically for timing analysis.

`-nobus`

This option directs LeonardoSpectrum to write busses in expanded form.

`-nocascades`

Does not map to cascades during technology mapping for Altera FLEX.

`-nocheck_complex_ios`

This switch allows the designer to turn off the Complex I/O Checker and Modifier in LeonardoSpectrum for Actel Act3 technology. The Complex I/O Checker and Modifier checks if a given design meets the constraints of the Actel Act3 architecture.

`-nocomplex_ios`

This switch directs LeonardoSpectrum to avoid mapping to complex I/O gates in the target technology. These are gates that combine I/O buffers and register logic. Applies only to Actel, Xilinx and Lucent 3000.

`-nocounter_extract`

This option disables automatic extraction of counters in VHDL and Verilog.

`-nodecoder_extract`

This option disables automatic extraction of decoders in VHDL and Verilog.

`-noglobal_symbol`

This option directs LeonardoSpectrum to process global set/reset when running with `-macro` option. When `-noglobal_symbol` is specified, a startup block is not instantiated.

`-noinfer_global_sr`

Disables detection of the global set or reset signal. Applies only to ORCA 2CA and 2TA and Xilinx 4000/E/EX/XL and 5200 target technologies.

`-nologic_rep`

Disables the use of logic replication to meet fanout limitations in the Actel FPGA architectures.

`-nolut_map`

Disables LUT mapping for lookup table based FPGA architectures Altera FLEX, ORCA, Xilinx 3000, 4000/E and 5200, and Lucent 3000.

`-nomap_global_bufs`

This option disables using global buffers for clocks and other global signals. Applies to Actel and Xilinx 3000, 4000/E and 5200 target technologies.

`-noopt=<list> | [{<gate_name>}]...[{<gate_name>}]`

When this option is used, all instances of the specified gate are marked NOOPT. The gates are not touched by the optimization algorithms. The gate appears as unchanged in the output design. If the gate does not exist in any of the input libraries, then a black box is created for this gate. Since information does not exist about the area, delay, input loading, or output drive of the gate, then reports on area and delay are not accurate and the output design may not be properly buffered. **Note:** This option can be used multiple times, or can accept a list of files as a parameter.

Example: design_name [`-noopt design_name`]

`-nopack_clbs`

This option disables packing Lookup Tables (LUTs) into CLBs. Only applies to Xilinx 4000/E, and 5200 target technologies.

`-nopld_xor_decomp`

This option prohibits LeonardoSpectrum from doing XOR decomposition. The XOR decomposition sometimes allows efficient mapping of a logic block into macrocells, if XOR decomposition results in fewer product terms than the original sum-of-product representation of that logic symbol. Applies to Xilinx XC9500 families.

`-noram_extract`

This option disables automatic extraction of RAMs from VHDL or Verilog.

`-nosdf_hierarchical_names`

This option treats all names with the divider character in the SDF file as names in a flat netlist.

`-notime_opt`

Do not run timing optimization.

`-notimespec_generate`

If used, TIMESPEC information from user constraints is not created. Applies to Xilinx 3000, 4000/E and 5200, and Lucent 3000 target technologies only.

`-notransformations`

Disables the conversion of latches, flip-flops, and I/O buffers into more primitive cells if these gates are not available in the target technology.

`-nowrite_eqn`

Disables writing of EQN Symbols (Xilinx) or LUT functions (Altera FLEX) in the output netlist. Applies only to Xilinx and Altera FLEX target technologies. Default is on. This option is not passed when the option is off.

`-nowrite_lut_binding`

Disables printing LUT binding (HMAP/FMAP/F5MAP) information in XNF/EDIF, if available. Applies only to ORCA, Xilinx 3000, 4000/E and 5200, and Lucent 3000 target technologies.

`-noxlx_fast_slew`

This option overrides setting outputs to FAST. If it is not used, all Output Buffers will be set to FAST only if slew rate attributes do not already exist. Only applies to: XC4000/A/E/EX/XL and XC5200 technologies.

`-noxlx_preserve_pins`

When used, this option inhibits the preservation of pin locations in the netlist when reading an XNF netlist.

`-noxnf_eqn`

Uses EQN symbols, instead of AND/OR symbols, when writing XNF. Default is on and the option is not passed unless user turns the option off.

`-num_crit_paths=<integer>`

The number of paths to report in the critical path report. The default number is 10.

`-optimize_cpu_limit=<integer>`

Specifies the CPU limit for the optimize command, in seconds. By default there is no limit.

`-output_format=<string>`

Specifies the format of the output design. This is optional if the output format can be determined from the extension of the output filename. Refer also to `-input_format`. Supported output file types and corresponding extensions are:

file_type	extension	description
XNF	.xnf	Xilinx Netlist Format
EDIF	.edf, .edif, .eds, .edn	EDIF netlist
VHDL	.vhd, .vhdl	VHDL
Verilog	.v, .vg, .vlg, .verilog	Verilog HDL

Both of the following specifications write an XNF file:

```
output_filename1 -output_format=xnf
output_filename2.xnf
```

`-package=<string>`

Specifies a specific package type for the output design. Applies to QuickLogic pASIC target technologies only.

`-pal_device`

Disables map to complex IOs for Actel/TI.

`-parallel_case`

When using the case statement in the Verilog input design, and case conditions are mutually exclusive, a multiplexer is often the preferred implementation (instead of priority encoding a state machine). This option specifies the multiplexer implementation. Also see `-full_case`.

`-part=<string>`

Specifies the target part when designing for Xilinx, ORCA or QuickLogic FPGAs. Just inserts the part number into the netlist; does not affect design optimization.

```
-pass=<list>
-nopass=<list> | [{<pass_number>}]...[{<pass_number>}]
```

`-pass` instructs LeonardoSpectrum to explicitly run only the specified Optimization/Technology Mapping pass, designated by `<pass_number>`. `-nopass` instructs LeonardoSpectrum to skip the designated pass. These options are only applicable when effort is `standard`. These options can be used multiple times, or can accept a list of pass numbers as a parameter.

`-preference_filename=<string>`

If specified, an ORCA preference file is produced.

`-preserve_dangling_net`

The `-preserve_dangling_net` option treats outputs with no load like external signals. By default, LeonardoSpectrum assumes all nets without fanout, and not explicitly declared external, to be useless. These nets are swept away. This option preserves such nets and applies only when the input format is XNF.

`-process=<string>`

Specifies the process variation used in delay calculations. Valid values are technology dependent. This option is used instead of speed grade for Actel, Altera, ORCA, Xilinx, and Lucent 3000 output options.

`-product=<ls1|ls2|ls3>`

Invokes a command line run of LeonardoSpectrum with Level 1 (ls1) or Level 2 (ls2) license instead of the default Level 3 (ls3) license.

`-propagate_clock_delay`

When this switch is turned on, delays are propagated through the clock tree along the paths to the clock input of the registers. This affects the arrival times at the outputs of registers, and also affects the arrival time at the end points. By default, this switch is turned off to represent an ideal clock.

`-report_brief|-report_full`

This option provides an area report and a report of arrival times, required times and slack at the 10 most critical or latest arriving end points. The `-report_full` option provides an area report and a critical path report in the summary file.

`-sdf_hier_separator=<string>`

This option specifies the separator character to use for hierarchical names in the standard delay format (SDF) writer. The default is /.

`-sdf_in=<string>`

Specifies the SDF input file to be read by LeonardoSpectrum.

`-sdf_names_style=<string> (vhdl|verilog|none)`

This option specifies the rename rules for the SDF writer. The default is vhdl.

`-sdf_out=<string>`

Specifies the file where LeonardoSpectrum writes delays in SDF format.

`-sdf_type=<string> (min|typ|max)`

Specifies type of delay for reading and writing SDF from LeonardoSpectrum. The default is `max`.

`-select_modgen=<string> (smallest|small|fast|fastest)`

Specifies the default mode for resolving modgen. Applies to VHDL and Verilog input formats only. The default is `auto` (`small` if optimizing for area, `fast` if optimizing for delay).

`-session_file=<string>`
`-nosession_file`

Override default session history filename. `-nosession_file` means do not generate a session history file.

`-simple_port_names`

Creates simple names for vector ports: `%s%d` instead of `%s(%d)`. For example, the name for bit one of a bus called abus is `abus1` instead of `abus(1)`.

`-source=<list> | [{<library_name>}]...[{<library_name>}]`

Specifies the source technology of the input design. To read a source file in a given technology, the corresponding technology library must be licensed and found in the search path. The search path for libraries is defined as the working directory and the $EXEMPLAR/lib directory, in that order. Libraries can also be specified with a complete path, relative to the working directory. This option can be used multiple times, or can accept a list of libraries as a parameter, for designs with multiple input technologies. **Note:** -source=<library_name> applies any time the input design has library specific cells instantiated in it. For pure RTL code, no source technology is required.

`-summary=<string>`
`-nosummary`

Specifies the file where the design summary report is written. The default name is `<output_filename>.sum`. Use `-nosummary` to prevent the summary file from being created.

`-target=<string>`

Specifies the target technology to map the optimized design to. To optimize for a given target technology, the corresponding technology library must be licensed and found in the search path.The search path for libraries is defined as the working directory and the $EXEMPLAR/lib directory, in that order. Libraries can also be specified with a complete path, relative to the working directory.

`-temp=<string>`

Specifies the operating temperature in celsius/centigrade. The timing information is derated for this operating temperature during delay computations. The value must be in the operating range of the target library.

`-tristate_map`

Enables conversion of tristate busses to combinational logic. This option is on by default for all Actel and Altera technologies.

`-use_qclk_bufs`

This option directs LeonardoSpectrum to use quadrant clocks for the Actel 3200dx architecture.

`-use_f6lut`

Enables mapping to 6-input LUTs during ORCA LUT mapping.

`-use_f5map`

Enables mapping to F5MAP for Xilinx 5200 technology.

`-verilog_file=<list>`

List of Verilog files to analyze before main input file.

`-verilog_wrapper=<string>`

Creates a Verilog wrapper file for the design. This may be used when busses are split in the synthesized netlist. By default busses are preserved in the output netlist and this option is not necessary. This option is only necessary if you use the `-nobus` option.

`-vhdl_file=<list>`

List of VHDL files to analyze before main input file.

`-vhdl_87|-vhdl_93`

Directs LeonardoSpectrum to read 1987 style VHDL. `-vhdl_87` is mutually exclusive with `-vhdl_93`. By default this option is off. The `-vhdl_93` option directs LeonardoSpectrum to read the 1993 style VHDL. `-vhdl_93` is mutually exclusive with `-vhdl_87`. `-vhdl_93` is the default.

`-vhdl_wrapper=<string>`

Creates a VHDL wrapper file for the design. This is useful when busses are split in the synthesized netlist. By default busses are preserved in the output netlist and this option is not necessary.

`-vhdl_write_87`

Uses VHDL ‘87 style syntax/semantics instead of VHDL ‘93 for writing VHDL.

`-vhdl_write_bit=<string>`

Specifies the type for the bit used in VHDL writer. The default is standard logic.

`-viewlogic_vhdl`

For VHDL synthesis, reads the ViewLogic pack1076 built-in package as the standard package.

`-voltage=<string>`

Specifies the operating voltage in volts. The timing information is derated for this operating voltage during delay computations. The value must be in the operating range of the target library.

`-wire_table=<string>`
`-nowire_table`

Allows the selection of a wire table at run time. One or more wire tables are specified in the library. Wire tables give an estimate of wire loads and delays as a function of fanout. These estimates can vary with the size of the module. With this option, a specific wire table can be selected. By default, the first wire table specified in the library is used. The `-nowire_table` option turns off the use of a wire table during delay calculations, which causes interconnect load and delays to be ignored.

`-wire_tree=<string> (best|balanced|worst)`

Sets the interconnect model. The default is `worst`.

Select `best` to get best case wire tree. This is the best case for interconnect delay. This is 0. The resistance of the wire is not a factor in the interconnect delay.

Select `balanced` to get a balanced wire tree. In this case, each segment of the wire resistance is equally distributed on each of the branches of the net.

Select `worst` (default) to get worst case wire tree. In this case, the full wire resistance is a factor in the delay, which creates high interconnect delay.

`-write_clb_packing`

Print CLB packing (HBLKNM) information in XNF/EDIF, if available. Applies to Xilinx 4000/E and 5200 target technologies only.

`-xlx_preserve_gsr`

Preserves the global set/reset (gsr) signal when using a Xilinx design as input to LeonardoSpectrum. This is only appropriate if the source technology is Xilinx, and the target technology is not Xilinx. Also, the GSR must appear explicitly in the input design.

`-xlx_preserve_gts`

Preserves the global tristate (gts) signal when using a Xilinx XC4000 design as input to LeonardoSpectrum. This is only appropriate if the source technology is Xilinx, and the target technology is not Xilinx.

Options for MAX+PLUS II Functionality

If you are running LeonardoSpectrum in batch mode, then MAX+PLUS II functionality is available through the following command line options.

`-maxplus2`

This option launches MAX+PLUS II in batch mode.

`-no_acf`

This option suppresses generation of acf file.

`-max_ta_reg`

This option analyzes register performance.

`-auto_fast_io`

This option enables auto fast IO.

`-auto_register_packing`

This option enables auto register packing.

`-max_exe=<executable>`

This option provides full path name of MAX+PLUS II executable file. This is needed if MAX+PLUS II is not in search path.

License Information

If options entered for a batch mode are incorrect, then LeonardoSpectrum attempts to enter the interactive command line shell mode. Since the interactive command line shell mode is for **Level 3** only, then a license error message comes up if you only have a Level 1 or 2 license.

Tcl Script Sourcing

After you create a Tcl script in a standard text editor, you can source your Tcl script from LeonardoSpectrum as follows:

- Interactive Command Line Shell (**Level 3**)
- GUI Menu Bar File -> Run Script
- Command Line with Path to LeonardoSpectrum

Note: The Exemplar history file is a Tcl script file that you can use after making the necessary edits.

Interactive Command Line Shell (*Level 3*)

Type the following syntax to source your Tcl script:

```
source <my_tcl_script>
```

GUI Menu Bar File -> Run Script

On the menu bar click on File -> Run Script. Type in your Tcl script name or click on the button and choose a Tcl script file. Your script file runs in the GUI Information window.

Command Line with Path to LeonardoSpectrum

Bring up your PC DOS or UNIX window. In the LeonardoSpectrum install area, locate where $EXEMPLAR points to the location of the software. Type the appropriate argument to source your Tcl script:

UNIX: `$EXEMPLAR/bin/spectrum -file <my_tcl_script>`

PC DOS: `$EXEMPLAR/bin/win32/spectrum -file <my_tcl_script>`

Note for Batch Mode:

UNIX: `$EXEMPLAR/bin/spectrum <input file> <output file> <target> <options>`

PC: `$EXEMPLAR/bin/win32/spectrum <input file> <output file> <target> <options>`

Power Tabs and Advanced Topics

The information presented in this chapter is for customizing your design.

- Input File Options
- Output File Options
- Optimize Options
- Advanced Technology, FPGA and ASIC
- Special Instructions for Mixing Design Languages
- Adding Libraries

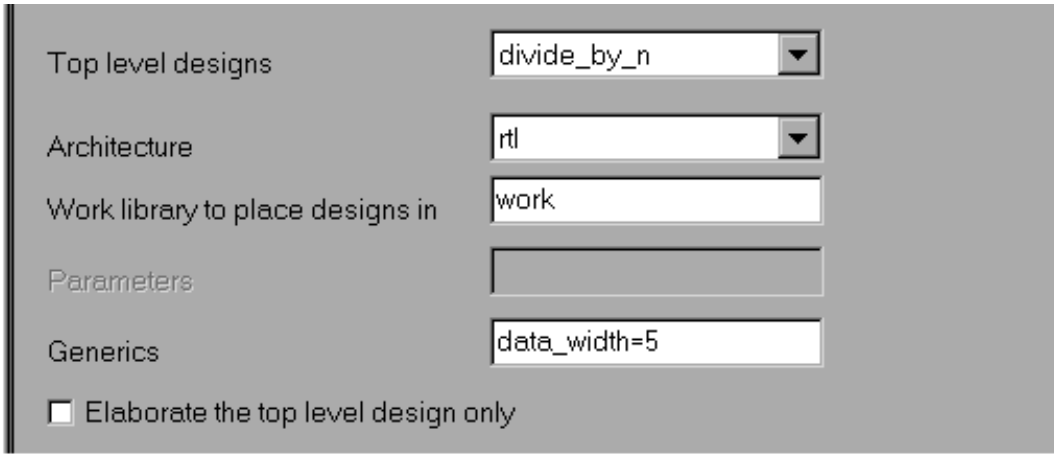
Input File Options

The Input tab includes the following power tabs:

- Elaborate, Table 7-1, **Level 3 only**
- VHDL, Table 7-2
- Verilog, Table 7-3
- XNF, Table 7-4
- EDIF, Table 7-5

*Table 7-1. Elaborate Input Options - **Level 3** Power Tab*

Option	Description
Set options and run the elaborate command on the input tab. Elaborate is used on files that are read in using the analyze command. Refer to the following example screen settings. **Note:** Before you can elaborate your design, you must select ■ Analyze Only on the Input tab. Also see rules for read, elaborate, and analyze at the end of this table.	



Top level designs	Use default name or select the top level design from pulldown. Hierarchical designs may have multiple files. The top level (entity) of your design is elaborated by default. You can also elaborate individual sub-entities, if any.
Architecture (VHDL only) **Note**: Specify the name in lower case only.	Use default architecture name or select from pulldown. An instance is composed of architecture and entity. For example, an entity consists of coded declarations that describe inputs and outputs and other net connections. Architecture is the function of an entity. For example, the function of an entity may be an adder, counter, multiplier, etc. Select the architecture of the top level entity. This entity is used as the top level of hierarchy in the input VHDL design. VHDL files are order dependent. VHDL requires that the top entity is at the bottom of the list. **Note:** Autotop detection takes care of ordering Verilog files.
Work library to place designs in	LeonardoSpectrum compiles your file when you click **Elaborate**. The files are saved in **work** by default.

*Table 7-1. Elaborate Input Options - **Level 3** Power Tab*

Option	Description
Parameters (gray)	The Parameters field allows you to set the value for parameter(s). Multiple parameters may be specified. <parameter>=<value>[<parameter>=<value>...] These are variable parameters like RAM, ROM.
Generics	Enter the memory size of a RAM, for example, **data_width=5**. In your source code, you may have a RAM memory size of 7. You can change the memory size here by typing the new value in Generics. <generic>=<value>[<generic>=<value>...]
☐ Elaborate the top level design only.	Select to elaborate only the top level of a multiple level design. If the box is not selected then the entire design is hierarchically elaborated. If your design is flat you may select to elaborate.

Note: Before compiling, LeonardoSpectrum checks (analyzes) for syntax errors in your source code. These errors, if any, are highlighted in the HDL editor window with a message.
Note: Elaborate and analyze are part of the synthesis process that results in a technology-independent netlist. Elaborate synthesizes your design to generic primitives and operators.

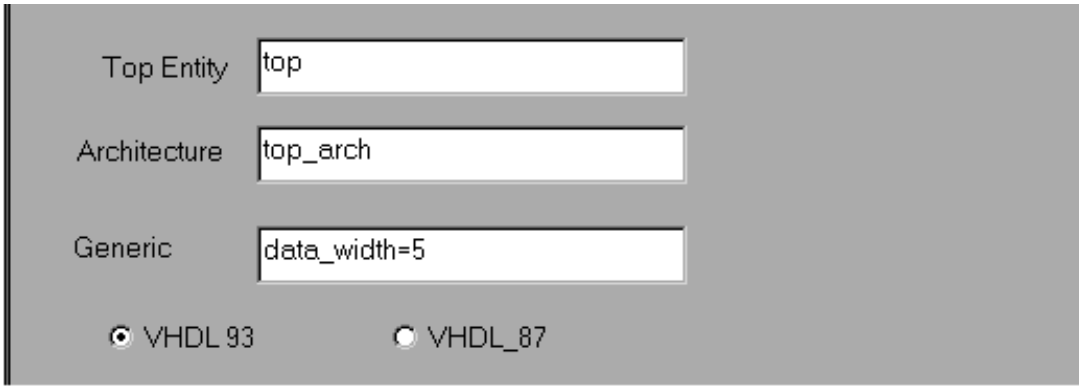
Rules for Read, Elaborate, Analyze:

- Read does both analyze and elaborate functions.
- Read f1, read f2,....read f10; you can always read files. LeonardoSpectrum checks for accuracy of EACH file. This may take considerable time.
- Instead of Read, you can analyze f1, analyze f2, ... analyze f10 and then elaborate. During elaborate, LeonardoSpectrum only needs to check accuracy once for all files. This may take less time than reading each file.
- You can also mix the functions by analyzing f1, then reading f2, and elaborating. You may use this if you want f1 to be a black box within f2 in your design.

Click **Elaborate** to implement options. Click **Help** for assistance.

Table 7-2. Input, VHDL Options - Power Tab

Options	Description
Use this power tab when your input design is in VHDL format. These options are used when reading a design from Input tab without selecting ☐Analyze Only. These options apply only to read. Refer to the example screen settings.	



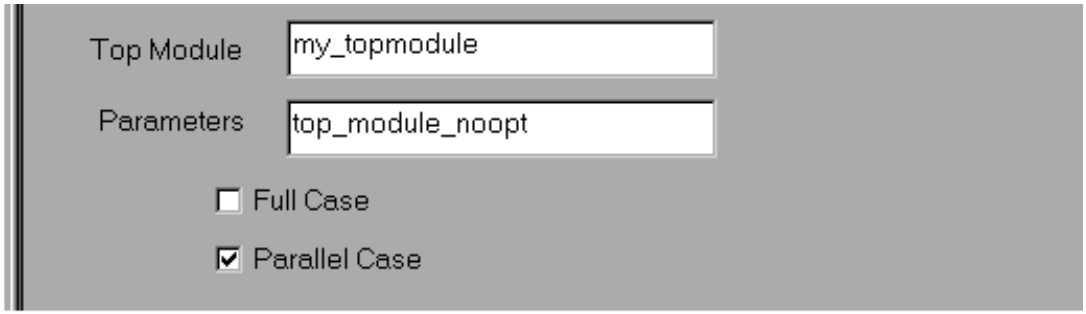
Top Entity:	Defines the top level of design hierarchy. If blank, the last entity read in is used. The top level is the last one found in the input file(s). Override this rule with the Entity option and specify the name of the top level entity.
Architecture: Specify name in lower case only.	When used with Entity option, defines the top level of design hierarchy. If blank, the last architecture that can be synthesized is used.
Generic (data_width=5)	Sets the value for specified generics in the format `<generic>=<value> [<generic>=<value>...]` Multiple generics may be specified in this format.
VHDL Style	●VHDL 93 ❍VHDL_87 either the 1993 or 1987 style of VHDL is read.

Click **Apply** to apply options. Click **Help** for assistance.

Table 7-3. Input, Verilog Options - Power Tab

Option	Description

Use this power tab when your input design is in Verilog format. LeonardoSpectrum applies autotop detection to Verilog files. Specify the name of the top module to override this rule. Refer to the following example settings.



Top Module: `my_top_module`

Parameters: top_module_noopt
The Parameters field allows you to set the value for parameter(s). Multiple parameters may be specified:
<parameter>=<value>[<parameter>=<value>...]
These are variable parameters like RAM, ROM.

☐ Full Case: This is a true full synthesis directive. Select to guarantee that the case statement is interpreted as a full case. If a default assignment was not used, then this option prevents the implementation of extraneous latches.

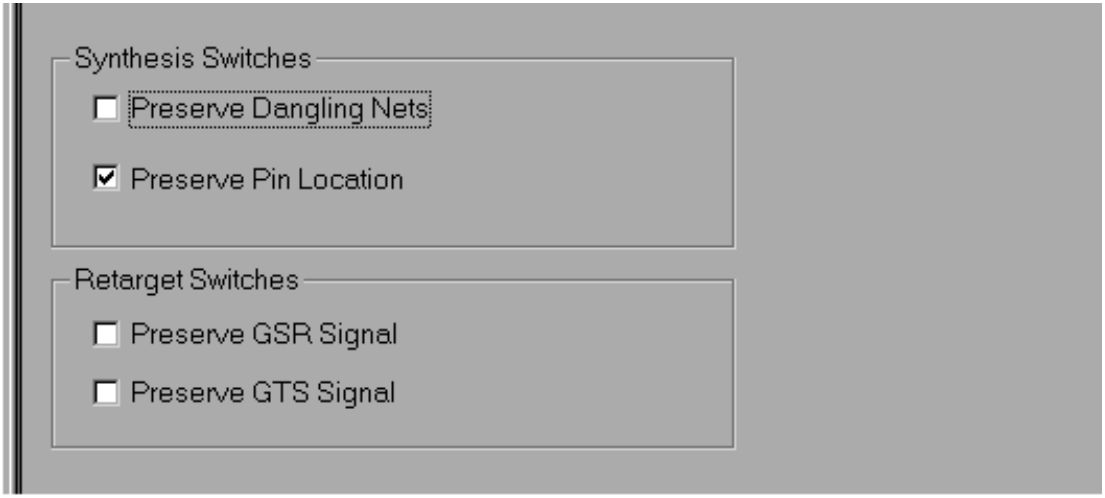
■ Parallel Case: This is a parallel synthesis directive. Select to guarantee that the case statement is parallel. A multiplexer may be the preferred implementation when case conditions are mutually exclusive. A multiplexer may also be the preferred implementation instead of priority encoding a state machine.

Click **Apply** to apply options. Click **Help** for assistance.

Table 7-4. Input, XNF Options - Power Tab

Option	Description

Use this power tab to set XNF specific options when your Xilinx design is in XNF format.



Synthesis Switches:
Preserve logic in XNF output netlist.

❒ Preserve Dangling Nets: Select to prevent a loadless net from being swept away during optimization. LeonardoSpectrum treats these nets like external signals. Otherwise, the nets are swept away.

■ Preserve Pin Location: Select to preserve pin location in output netlist. Disable preservation of pin location when performing Xilinx->Xilinx optimization.

Retarget Switches:

❒ Preserve GSR Signal: Select to preserve GSR signal during optimization. This preserves the global three state (gts) signal when using XC4000 design as input to LeonardoSpectrum. This is used when the target technology is a technology other than Xilinx.

❒ Preserve GTS Signal: Select to bypass GTS signal during optimization.

Click **Apply** to apply options. Click **Help** for assistance.

Table 7-5. Input, EDIF Options - Power Tab

Option	Description

Use this power tab when your input design is in EDIF format.

Design
This is the top level of design to read.

`my_edif_design`

LeonardoSpectrum expects the design name to be supplied in the EDIF file using the EDIF “design” construct. Override this rule by specifying the name of the root (or top level) cell.

Click **Apply** to apply options. Click **Help** for assistance.

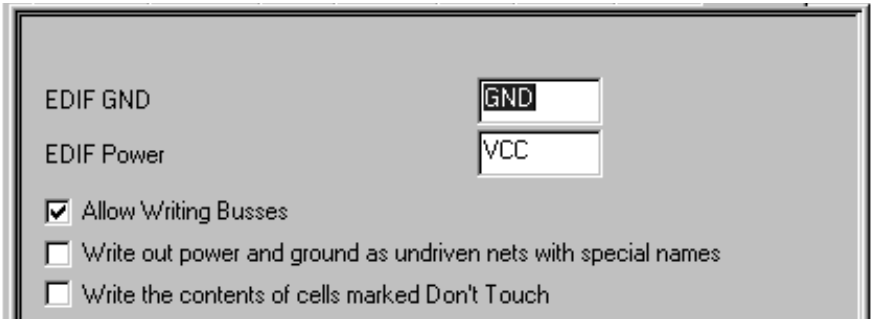
Output File Options

The Output tab includes the following power tabs:

- EDIF, Table 7-6
- SDF, Table 7-7
- Verilog, Table 7-8
- VHDL, Table 7-9

Table 7-6. EDIF Out Options - Power Tab

Option	Description
Set EDIF specific options before writing out the EDIF output format.	


EDIF GND
GND
EDIF Power
VCC
Allow Writing Busses
Write out power and ground as undriven nets with special names
Write the contents of cells marked Don't Touch

EDIF GND — Accept default GND or type in your choice. This is your special name for ground nets. Be sure to select ■Write out power and ground as undriven nets with special names.

EDIF Power — Accept default VCC or type in your choice. This is your special name for EDIF power. Be sure to select ■Write out power and ground as undriven nets with special names.

■Allow Writing Busses: Before V1998.2, LeonardoSpectrum split busses. For example, A0 and A1 were split into individual bits for writing busses. Now you can write A+B=sum to indicate that A bus + B bus is the sum.

❒Write out power and ground as undriven nets with special names: Enter your names for ground and power.

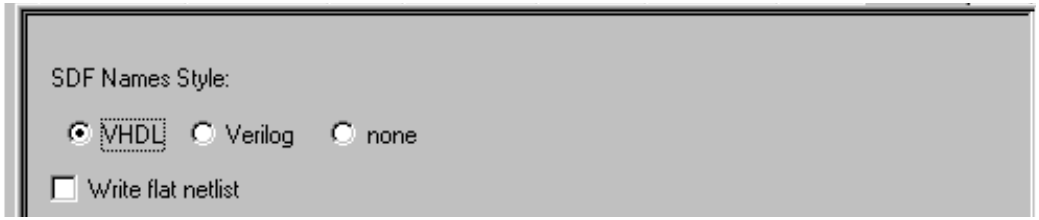
❒Write the contents of cells marked Don’t Touch. Objects marked with `dont_touch` are not optimized or unmapped. In contrast to `noopt`, `dont_touch` prevents optimization of the lower levels of hierarchy and leaf instances.

Click **Apply** to apply options. Click **Help** for assistance.

Table 7-7. SDF Out Options - Power Tab

Option	Description

Set your specific SDF options before writing out an SDF file. SDF is not a netlist; SDF is a format.



SDF Names Style:	SDF is a netlist format which derives a style from VHDL or Verilog.
●VHDL	The output file is in a SDF netlist format with a VHDL style.
❍Verilog	The output file is in a SDF netlist format with a Verilog style.
❍none	The output file is in a SDF netlist format with the default VHDL style.
❒Write flat netlist	The hierarchy of your design is flattened unless this choice is selected.

Click **Apply** to apply options. Click **Help** for assistance.

Table 7-8. Verilog Out Options - Power Tab

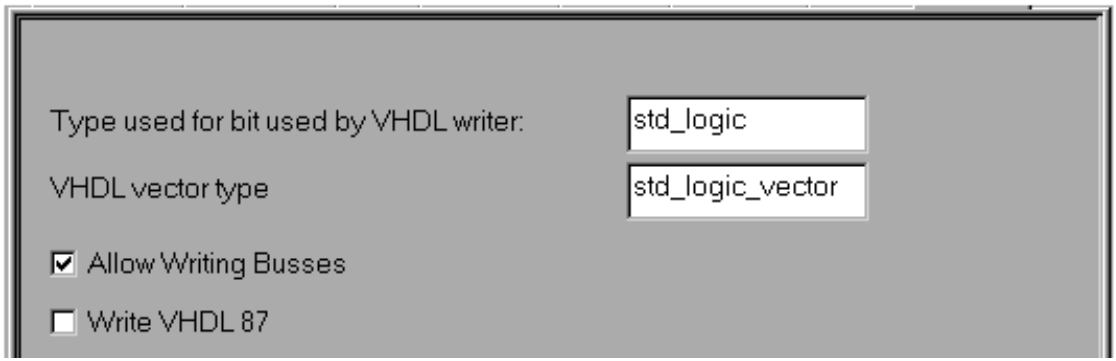
Option	Description

■ Allow Writing Busses: Before V1998.2, LeonardoSpectrum split busses. For example, A0 and A1 were split into individual bits for writing busses. Now you can write A+B=sum to indicate that A bus + B bus is the sum.

Click **Apply** to apply options. Click **Help** for assistance.

Table 7-9. VHDL Output Options - Power Tab

Option	Description
Type used for bit by VHDL writer:	std_logic - This type allows a choice of nine values (0, 1, X, L, H, W, U, -, Z). Other bit choices allow 0, 1 only.



VHDL Vector Type	std_logic_vector Specify the type for the bit-vector used in VHDL writer.

■ Allow Writing Busses: - Before V1998.2, LeonardoSpectrum split busses. For example, A0 and A1 were split into individual bits for writing busses. Now you can write A+B=sum to indicate that A bus + B bus is the sum.

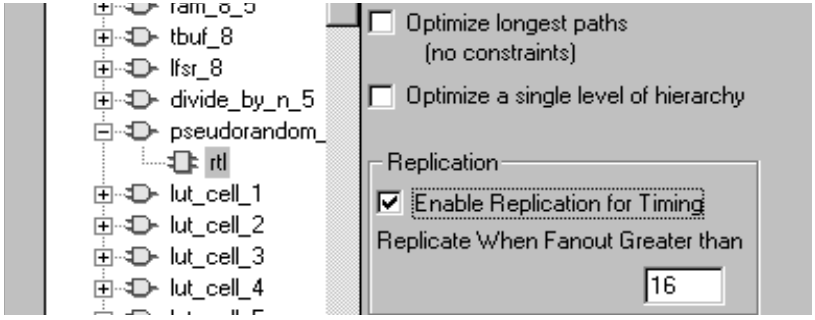
☐ Write VHDL_87: This directs LeonardoSpectrum to read 1987 style VHDL instead of 1993 style VHDL.

Click **Apply** to apply options. Click **Help** for assistance.

Optimize Options

Table 7-10. Optimize Timing (**Level 3**) - Power Tab

Field, Window, and Button	Description
Select design to optimize:	Browse through the tree in the interactive window and click on a design. This tree was created when you read in the design. Selecting from the tree allows you to perform various timing optimizations in the design.
Current Path: work.pseudorandom_8.rtl	The design selected from the tree in the interactive window is displayed in the Current Path.
❑Optimize longest paths (no constraints)	Optimizations are concentrated on paths in the design that violate timing.
❑Optimize a single level of hierarchy.	Select to limit optimization to one level of your design. Timing optimization is performed only at top level of hierarchy. If this option is not selected, then the all views in the design are optimized.
■Enable Replication for Timing. Replicate When Fanout Greater than: 16 (for example). **Note:** Grayed out for ASIC.	This is for All Xilinx (not Virtex) and all Altera FLEX technologies. For Xilinx the F and H function generators are replicated for a number >= to the entered fanout value: 16 (for example). For Altera FLEX the LUTs are replicated.



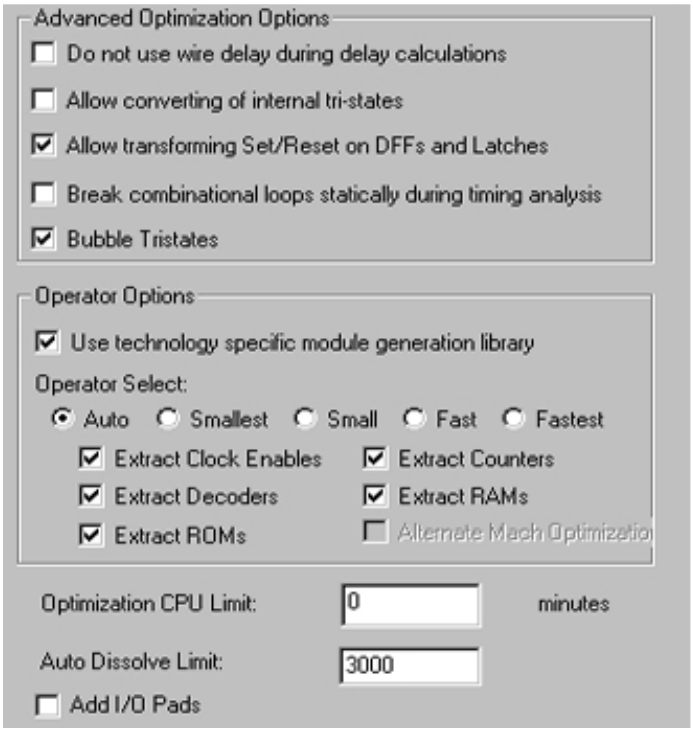
Optimize: Click to start optimizing.

Note: The optimize_timing command is a constraint driven timing optimizer that is driven from static timing analysis results. The optimizations are concentrated on paths in the design that violate timing. optimize_timing examines the most critical paths and attempts to improve the arrival_time at the end of each path. If you apply constraints, the command is more effective.

Table 7-11. Advanced Optimization Options - FPGA/CPLD and ASIC - Power Tab

Option	Description

Advanced Optimization Options: Select power tab options on Optimize tab before you optimize your design.



❑Do not use wire delay during delay calculations. (Variable: wire_table FALSE). Select when interconnect delays are ignored. The use of a wire table during delay calculations is disabled.

❑Allow converting internal tri-states. (Variable: tristate_map FALSE). Select to allow the conversion of internal tristates to combinational logic that matches the target technology.

■Allow transforming Set/Reset on DFFs and Latches. (Variable: transformation FALSE). Select to allow transformations to match the target technology.

❑Break combinational loops statically during timing analysis. (Variable: delay_break_loops FALSE). Select to allow combinational loops to be broken statically for timing analysis and critical path reporting. The default is dynamic analysis of combinational loops. This option speeds up timing analysis, optimization and critical path reporting when dynamic analysis takes too much time. However, timing analysis is not accurate when this option is used and the design has combinational loops. *Table 7-10 continued...*

...Table 7-11 continued

■ Bubble Tristates: (Grayed out for ASIC) Use these rules:

- If tristates are not in common levels, then by selecting ■ Bubble Tristates, the tristates bubble up to the common top level.
- If tristates are in a common level and feeding the output port, then by selecting ■ Bubble Tristates, the tristates bubble up to the top primary output port. This also occurs if tristates are not in a common level. **Note:** Refer to Command Reference, Utilities chapter for more information.

Operator Options: (For VHDL and Verilog input formats only.)

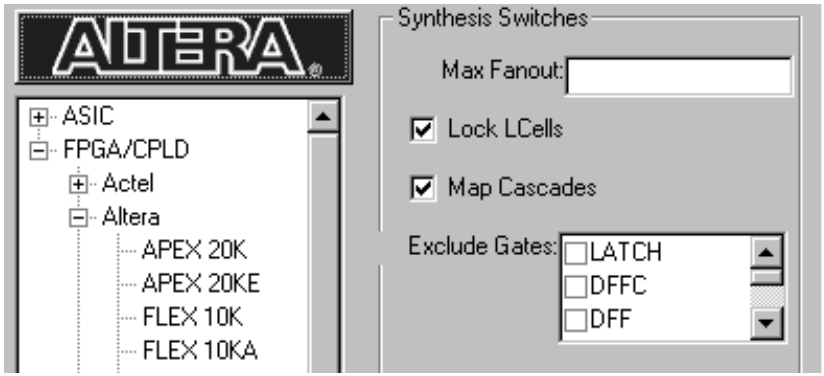
■Use technology specific module generation library. ❑If the box is not checked, then a default internal module generation routine is used.

Operator select: (For VHDL and Verilog formats only)	●Auto: Picks smallest if in area mode; picks fastest if optimization in delay mode.
	❍Smallest: Picks the most compact implementation available.
	❍Small: Picks a compact implementation.
	❍Fast: Picks a fast implementation.
	❍Fastest: Picks the fastest implementation available.
■ Extract Clock Enables:	Map to clock-enable flip-flops from VHDL and Verilog.
■ Extract Decoders:	Controls automatic extraction of decoders in VHDL and Verilog.
■ Extract ROMs	Controls automatic extraction of ROMs in VHDL and Verilog.
■ Extract Counters:	Controls automatic extraction of counters in VHDL and Verilog.
■ Extract RAMs	Controls automatic extraction of RAMs in VHDL and Verilog.
■ Alternate Mach Optimizatio	**Lattice/Vantis only**: Applies alternate synthesis heuristics. This flow may produce better results on smaller designs.
Optimization CPU Limit: min	Time needed for optimization algorithms to complete. 0 minutes, no time limit.
Auto Dissolve Limit:	3000 gates, default FPGA/CPLD
ASIC Auto Dissolve Limit:	30 gates, default ASIC (ASIC available if technology is ASIC.)
❑Add I/O Pads	If ❑I/O Pads is not selected, then LeonardoSpectrum runs the optimization in the macro mode. If ■I/O Pads is selected, then the optimization runs in the chip mode and pads are inserted in your design.

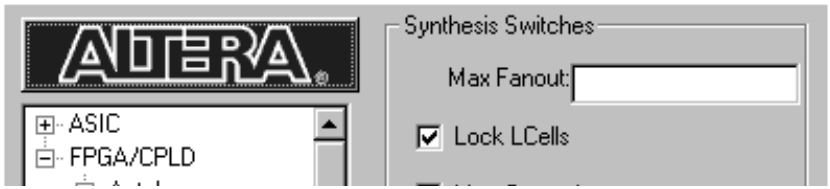
Click **Apply** to apply options. Click **Help** for assistance.
Note: Your original design is copied with an RTL extension, my_design_RTL. The original design is optimized, while the copy is retained as a record.

Advanced Technology FPGA

Table 7-12. Advanced Settings Power Tab for Altera FLEX 6K Technology (Example).

GUI Option	Option on/off	Interactive Command Line Shell	Batch Mode Option
■ Map Cascades	on	default (true)	default (true)
	off	`set flex_use_cascades false`	`-nocascades`
■ Lock LCells	on	default (false)	default (false)
	off	`set dont_lock_lcells true`	`-dont_lock_lcells`
			
☐ Lock LCells	If Lock LCells is not selected, then this directs LeonardoSpectrum not to force LCell buffers in the output EDIF netlist. You can then use MAX+PLUS II to map the combinational logic into LCells. FAST is the recommended setting for MAX+PLUS II `GLOBAL_PROJECT_SYNTHESIS_STYLE`.		
■ Map Cascades	By default this option is selected. LeonardoSpectrum then maps to cascade gates where applicable.		
Exclude Gates Scroll through the Exclude Gates list and highlight the gate(s) as needed. The listed Altera FLEX 6K gates are: Latch, DFFC, FF, DFFP, DFFE, TRI, TRIBUF, CBUF, SCLK, Global, INBUF, OUTBUF, BDBUF, TFF, TFFC, TFFP The selected gate(s) are excluded from the library when your design is mapped to the technology. In addition, excluded gates are not saved as part of a Project.			
Advanced Technology continued.....			

.

<table>
<tr><td colspan="2">continued... Advanced Technology</td></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td>Max Fanout:</td><td>Use the Max Fanout field on the GUI to override the default max fanout load specified in the library. However, a synthesized netlist with high fanout nets may be a problem for the place and route tool. The place and route tool usually splits the net arbitrarily. High fanout nets can cause significant delays on wires and become unroutable. On a critical path, high fanout nets can cause significant delays in a single net segment and cause the timing constraints to fail.
To eliminate the need for splitting of the net by the place and route tool, the synthesis tool must maintain a reasonable number of fanouts for a net. LeonardoSpectrum tries to maintain reasonable fanout limits for each target technology. Default fanout limits are derived from the synthesis library.
Note: The LUT buffering and replication is supported for the Altera FLEX 6/8/10 and 10KA/KE/KB technologies.
General Rule
LeonardoSpectrum maintains reasonable fanouts by replicating the driver which results in net splitting. If replication is not possible, the signal is buffered. The buffering of high fanout primary input signals is an example. Buffering the signal causes the wire to be slower by adding intrinsic delays.
User Switches
On the interactive command line shell type:
<code>set lut_max_fanout <integer></code>

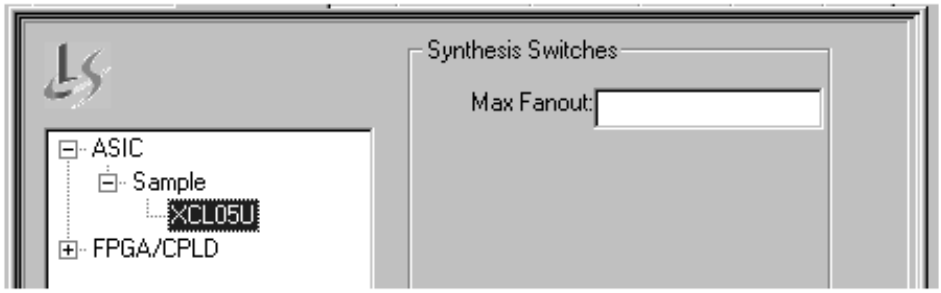
On specific nets, you can set an attribute to control the max_fanout value:
<code>set_attribute -net <net_name> -name lut_max_fanout -value <int></code>
Note: Setting this attribute takes precedence over any global fanout specifications.</td></tr>
<tr><td colspan="2">Click Load Library</td></tr>
</table>

Advanced Technology ASIC

Table 7-13. Advanced Settings Power Tab for Altera FLEX 6K Technology.

GUI Option	Description

Max Fanout	This is a maximum number of technology gates being driven by an instance. LeonardoSpectrum automatically applies a value for Max Fanout. Use the Max Fanout field on the GUI to override the default max fanout load specified in the library. However, a synthesized netlist with high fanout nets may be a problem for the place and route tool. The place and route tool usually splits the net arbitrarily. High fanout nets can cause significant delays on wires and become unroutable. On a critical path, high fanout nets can cause significant delays in a single net segment and cause the timing constraints to fail.
	To eliminate the need for splitting of the net by the place and route tool, the synthesis tool must maintain a reasonable number of fanouts for a net. LeonardoSpectrum tries to maintain reasonable fanout limits for each target technology.
	General Rule
	LeonardoSpectrum maintains reasonable fanouts by replicating the driver which results in net splitting. If replication is not possible, the signal is buffered. The buffering of high fanout primary input signals is an example. Buffering the signal causes the wire to be slower by adding intrinsic delays.
	User Switches
	On the interactive command line shell type:
	`set lut_max_fanout <integer>`
	On specific nets, the user can set an attribute to control the max_fanout value:
	`set_attribute -net <net_name> -name lut_max_fanout -value <int>`
	Note: Setting this attribute takes precedence over any global fanout specifications.

	
Click **Load Library**.	

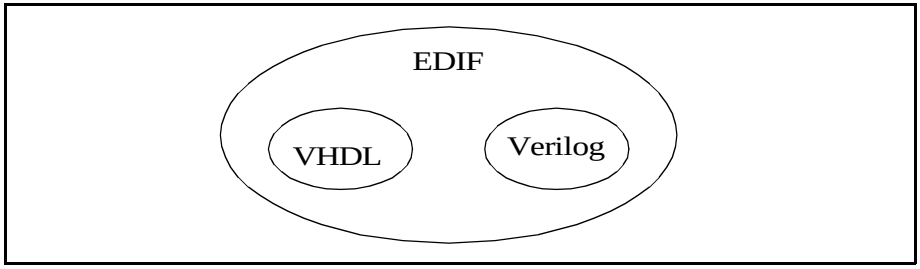
Special Instructions for Mixing Design Languages

LeonardoSpectrum provides two databases: (1) hdl database and (2) hierarchy database. By using a bottom up approach, you can read in a hierarchical design written in different languages. The idea is to first read in the lower levels of hierarchy into the hierarchy database and then to read the top level. The following two examples are for mixing Verilog, VHDL, and EDIF design languages.

Example (1) VHDL and Verilog instantiated separately in EDIF. Refer to Figure 7-1, and use these example steps as a guide for instantiating individual low-level blocks from VHDL and Verilog into a top-level EDIF design.

1. Read in a Verilog design into a top-level EDIF design.
2. Next, read in a VHDL design into a top-level EDIF design.
3. Finally, read in the EDIF design which now includes the individual low-level Verilog and VHDL blocks. This top-level design is now stored in the hierarchy and hdl database.

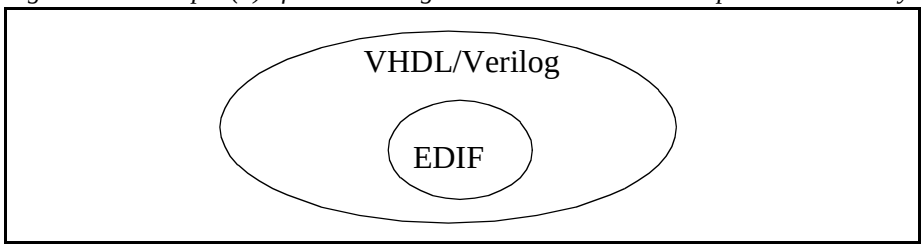
Figure 7-1. Example (1) of Instantiating Low-Level Blocks in a Top Level Hierarchy



Example (2) EDIF instantiated in VHDL/Verilog. Refer to Figure 7-2 and use these example steps as a guide for instantiating low level blocks from EDIF into a top level VHDL/Verilog design.

1. Read in a low level EDIF design.
2. Finally, read in the VHDL/Verilog design which now includes the low-level EDIF block. This top-level design is now stored in the hierarchy and hdl database.

Figure 7-2. Example (2) of Instantiating a Low-Level Block in a Top Level Hierarchy

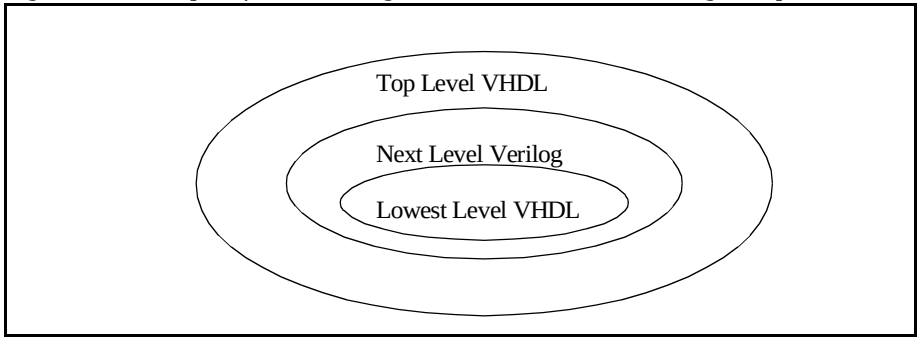


Clean Up Language Database - VHDL and Verilog

If you want to instantiate a VHDL or Verilog design into a Verilog or VHDL design, then you need to clean up the hdl database after reading in the lower hierarchy and before reading in the higher level of hierarchy. Refer to Figure 7-3 and use these example steps to clean up the hdl database:

1. Read in the lowest level VHDL. For example:
   ```
   read lowest.vhd
   ```
2. Clean up language database. Type:
   ```
   remove -hdl .
   ```
3. Read in the next level Verilog design. For example:
   ```
   read next.v
   ```
4. Clean up language database. Type:
   ```
   remove -hdl .
   ```
5. Finally, read in top level VHDL design with the low level, instantiated VHDL and Verilog designs. For example:
   ```
   read top.vhd
   ```

Figure 7-3. Example of Instantiating Low Level VHDL and Verilog in Top Level VHDL



*Adding a Library to the GUI (**Level 3**)*

Follow these steps to add additional libraries for your FPGA or ASIC technologies.

1. Use a text editor to bring up the `devices.ini` file. This file is located in the `$EXEMPLAR/lib` directory. Check for a listing of the FPGA or ASIC library in the Exemplar library subdirectory. If the library is listed, continue to step 2. Otherwise, complete the items in Table 7-12 and then continue to step 2.

Table 7-12. Definitions for `devices.ini` file

File Line: “Manufacturer”, “Family”, “Library”, “none”, Type”, “I/O”, “ “ , “ “, “ “ where,	
Manufacturer	This is the name that appears at the top level in the technology tree.
Family	This is the name that appears under the manufacturer’s name in the technology tree.
Library	This is the name of the library file. **Note**: There is no `*.syn` extension.
Type	Enter either FPGA for an FPGA library, or ASIC for an ASIC library.
I/O	Enter SOURCE if this library can be used only as a source technology. Enter DEST if this library can be used only as a target technology. Enter BOTH if this library can be used as both a source and target technology.
Example ASIC: “sample”, “SAMPLE”, “XCL05U”, “none”, “ASIC”, “BOTH”, “ “, “ “, “ “	
Refer to your `devices.ini` file for additional examples.	

2. Copy the `<tech>.syn` file into the `$EXEMPLAR/lib` directory. For an ASIC library type:

UNIX: `cp sample.syn $EXEMPLAR/lib`

Note: `$EXEMPLAR` (upper case) is allowed in the UNIX file. Refer also to Special Instructions for Adding Libraries at the end of this chapter.

Windows: use the Windows explorer to copy the library file to the library subdirectroy of LeonardoSpectrum.

Note: If the `<tech>.vhd` file exists in the design kit, then copy `<tech>.vhd` to the `$EXEMPLAR/data/modgen` directory.

Note: Libraries listed in `devices.ini` file that do not have the `*.syn` extension, are not available in the GUI.

Adding a Library to the GUI

Follow these steps to add additional libraries for your FPGA technologies.

1. Use a text editor to bring up the `devices.ini` file. This file is located in the `$EXEMPLAR/lib` directory. Check for a listing of a FPGA library in the Exemplar library subdirectory. If the library is listed continue to step 2. Otherwise, complete the items in Table 7-13 and then continue to step 2.

Table 7-13. Definitions for `devices.ini` file

File Line: "Manufacturer", "Family", "Library", "none", Type", "I/O", " " , " ", " " where,	
Manufacturer	This is the name that appears at the top level in the technology tree.
Family	This is the name that appears under the manufacturer's name in the technology tree.
Library	This is the name of the library file. **Note**: There is no `*.syn` extension.
Type	Enter an FPGA library.
I/O	Enter SOURCE if this library can be used only as a source technology. Enter DEST if this library can be used only as a target technology. Enter BOTH if this library can be used as both a source and target technology.
Example FPGA: "altera", "altera", "flex6", "none", "FPGA", "BOTH", " ", " ", " "	
Refer to your `devices.ini` file for additional examples.	

2. Copy the `<tech>.syn` file into the `$EXEMPLAR/lib` directory. For an FPGA library type:

UNIX: `cp flex6.syn $EXEMPLAR/lib`

Note: `$EXEMPLAR` (upper case) is allowed in the UNIX file. Refer also to Special Instructions for Adding Libraries at the end of this chapter.

Windows: use the Windows explorer to copy the library file to the library subdirectory of LeonardoSpectrum.

Note: If the `<tech>.vhd` file exists in the design kit, then copy `<tech>.vhd` to the `$EXEMPLAR/data/modgen` directory.

Note: Libraries listed in `devices.ini` file that do not have the `*.syn` extension, are not available in the GUI.

Special Instructions for Adding Libraries

UNIX is case sensitive with respect to the fields of “Library” and “Symbol Library”. For example, if the library is named “Fujitsu50K”, then the library field should be in the same mix of upper and lower case characters. The fields of “Manufacturer” and “Family” are not case sensitive.

The library names in the “Library” and “Symbol Library” fields should be entered without a “.syn” or “.sglib” suffix. The libraries from Exemplar are named according to the convention and do not require a suffix.

LeonardoInsight 8

A design analysis environment is integrated in LeonardoSpectrum - LeonardoInsight. LeonardoInsight provides the tools for interactive exploration of your synthesized design. For example, with the schematic viewer you can view the entire design, the critical path, or cone fragments. Moreover, cross probing is available to link your HDL source code, schematic design, and design browser for quick debugging. Cross probing is also available between schematics generated in Renoir and in LeonardoSpectrum.

LeonardoInsight includes:

- Cross Probing
- Schematic Viewer
- Schematic Fragments for Fanin and Fanout Cones
- Schematic Fragments for Critical Paths
- Hierarchical Design Browser

This chapter presents:

- Cross Probing
- Schematic Fragments
- Schematic Viewer Description
- Design Browser Description
- Adding a Technology Specific Symbol Library

Cross Probing

The design database supports cross probing between the schematic viewer, design browser, and HDLInventor source code editor as follows:

- The design browser is always synchronized with the database.
- A valid schematic in the schematic viewer is synchronized with the design browser by default.
- Highlighted HDL source code line(s) cross probe with the design browser and schematic viewer if line number and file name information is in the database for highlighted line(s).

Note: Highlighted objects may not always be immediately visible in the design browser or schematic viewer. In addition, some source code lines may not initiate cross probing after optimization.

Note: On the toolbar, toggle cross probe icon on and off.

Note: Since clicking on the main window brings that window forward, avoid overlapping the main window with the schematic viewer during cross probe.

Note: If your schematic viewer is gray with the **Warning: Invalid schematic**, then your schematic is invalid. A schematic becomes invalid when the database loses correspondence with the displayed schematic; the database netlist has been changed. On the schematic viewer toolbar, click on the Update Schematic button at the far right to reestablish correspondence between schematic and database. Refer ahead to Schematic Viewer Description in this chapter.

This cross probing section is divided as follows:

- Cross Probing between Renoir and LeonardoSpectrum
- HDL Constructs that Initiate Cross Probing
- Cross Probing from HDL Source Code
- Cross Probing to HDL Source Code
- Cross Probing between RTL and Gate-Level Schematic

Cross Probing Between Renoir and LeonardoSpectrum

Open a schematic in LeonardoSpectrum that is created in Renoir. On the Tools pulldown menu, check Renoir Crossprobe. Open the schematic and double click on an instance, for example. Renoir opens with the same instance highlighted.

HDL Constructs that Initiate Cross Probing

In addition to cross probing between a schematic and the design browser, LeonardoSpectrum introduces the unique ability to cross probe between the HDL source code, design browser, and schematic. The hierarchy database maintains line numbers and file names to identify certain HDL constructs in your source code. If you click on one of the following constructs in your source code, then cross probing should occur.

Note: This HDL discussion is intended to be generic for both Verilog and VHDL.

Declaration

A cell has the line number and file name of the entity. A view has the line number and file name of the architecture. Flip flops and tristate buffers get line numbers from the corresponding declaration of the signal/reg declaration.

Expressions

Expressions like arithmetic, boolean, and logical are prevalent in design files. The cross probing line number and file name information is only for user written expressions that create an operator and not for expressions derived from other standard packages.

Procedure Call

The logic created for a procedure call has the line numbers for the body of the procedure call.

Instantiation

The database contains instance line number and file name for component entity and configuration instantiation.

If Statements and Conditional Signal Assignment

Logic created from boolean statements has the line number and file name information of the if statement. Cross probing information for if statements and conditional signal assignments is created identically. For example, instances are created for an if statement. Instances then have the line number and file name information on the if statement.

```
if (bool-expr0)...
elsif (bool-expr1)...
elsif (bool-expr2)...
elsif (bool-expr3)...
else statements...
end if;
```

Case Statement and Selected Signal Assignment

Cross probing information for case statements and selected signal assignments is created identically. For example, logic created by a case statement has the line number and file name of the case statement.

```
case (expression)
when “000” <=
statements...

when “001” <=
statements...

when “011”|”100” <=
statements...

others <=
statements...
end case;
```

Variable Array Indexing

Instances created by variable array indexing have the line number and file name of the variable array index. For example, y <= A(i);

Cross Probing from HDL Source Code to Schematic Viewer

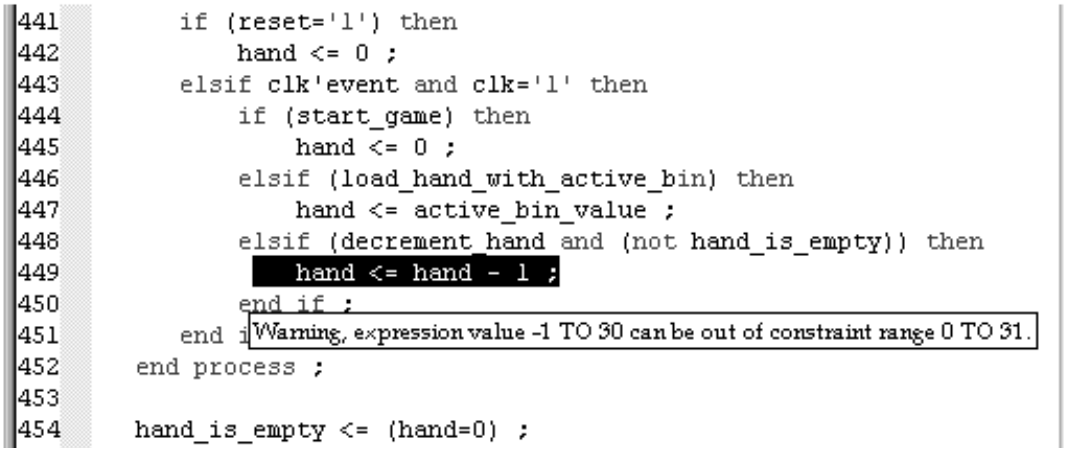
The following steps assume you have read your design successfully and the design browser and HDLInventor are in the main window, while the schematic viewer is not overlapping the main window. The following three screens 8-1, 8-2, and 8-3 are examples of cross probing to locate modgen_19 for the demo design, Mancala. Use these steps to recreate the demo example or a similar example:

Note: Assume left mouse button is used unless right mouse button (RMB) is indicated.

1. Start by bringing up the HDL source code editor with the example Mancala demo running and highlight Line 449, hand <= hand -1;. As shown, you are prompted with a popup message. Line 449 is an expression with line number and file name information in the database for the created logic. Refer to Screen 8-1, Part of HDL Source Code with Cross Probe. **Note:** Some lines may not initiate cross probing. Refer again to HDL Constructs that Initiate Cross Probing.

2. Next, double click on the HDL source code line to start cross probe. You can also click on **Analysis** on the menu bar and then click on **Trace to Hierarchy**. **Analysis** is only available when the HDL source code window is active. **Note:** Cross probing messages may appear in the status bar at the bottom of the HDLInventor.

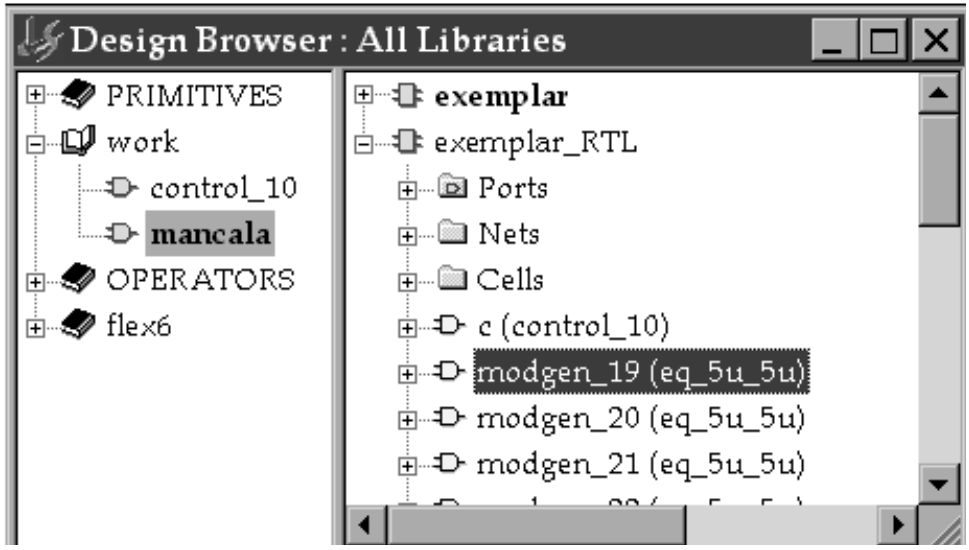
Screen 8-1. Part of HDL Source Code with Cross Probe


441 if (reset='1') then
442 hand <= 0 ;
443 elsif clk'event and clk='1' then
444 if (start_game) then
445 hand <= 0 ;
446 elsif (load_hand_with_active_bin) then
447 hand <= active_bin_value ;
448 elsif (decrement_hand and (not hand_is_empty)) then
449 hand <= hand - 1 ;
450 end if ;
451 end i
Warning, expression value -1 TO 30 can be out of constraint range 0 TO 31.
452 end process ;
453
454 hand_is_empty <= (hand=0) ;

Note: Example line numbers (..., 449, 450, 451, ...) on actual display may differ from line numbers on screen shot. Popup messages may also differ.

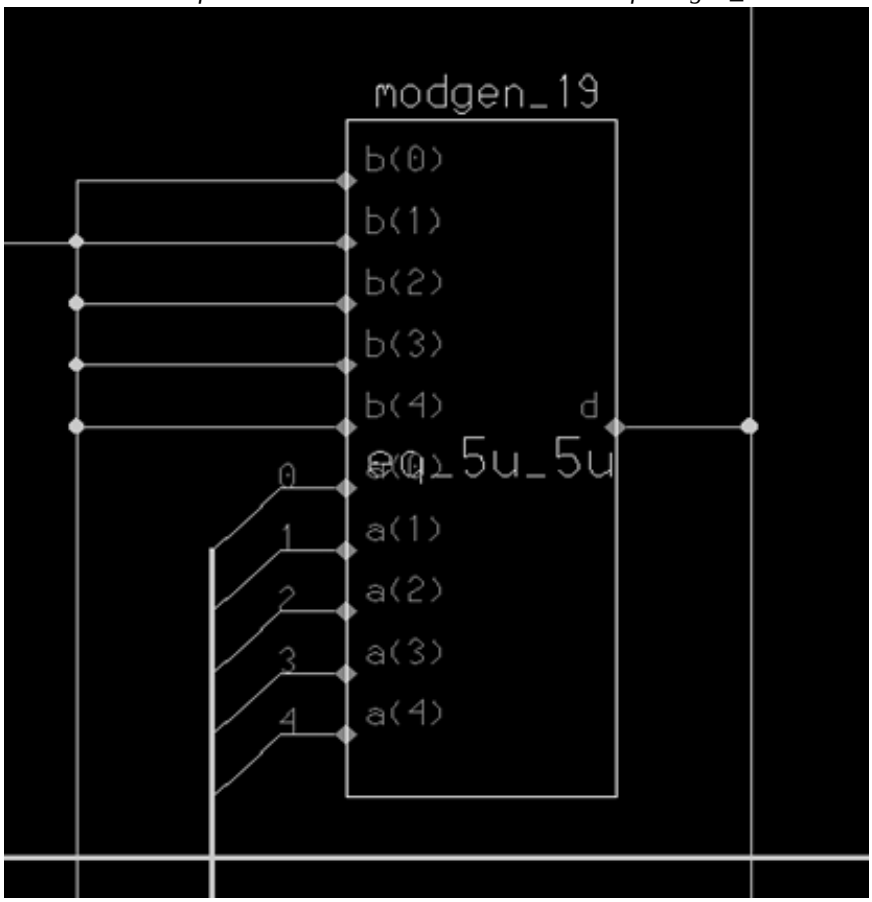
3. Expand the design hierarchy in the design browser to locate the object for the highlighted Line 449 in the HDL source code. On the design browser click to expand the tree and locate highlighted modgen_19. Refer to Screen 8-2.

Screen 8-2. Part of Design Browser Cross Probe



4. On the schematic viewer you may need to zoom in or out to locate the highlighted modgen_19. For convenience, only a portion of the zoomed in schematic viewer window is shown in Screen 8-3.

Screen 8-3. Part of Zoomed Cross Probe Schematic View of modgen_19



Cross Probing to HDL Source Code

Use these steps:

1. Select an object in the schematic viewer. The object will be highlighted in the design browser by default.
2. Scroll through your HDL source code to locate the highlighted line. If the schematic viewer object is identified with cross probe line number and file name information in the database, then it will be highlighted in the HDL source code.

Cross Probing Between RTL and Gate Level Schematic

Use these steps to correlate your original design (copy of original RTL primitives) with the gate-level technology cells in the design database. **Note**: Refer to Design Browser Description in this chapter.

1. Bring up an RTL schematic view together with the synthesized gate-level view.
2. Click to highlight object(s) in the RTL view and check for the presence of the object(s) in the gate-level view. Some original RTL objects may not be present after gate-level generation.

Schematic Fragments

The following special types of schematics are presented:

- Schematic Fragments for Critical Path Analysis
- Schematic Cones for Fanin or Fanout Logic from Selected Nodes

Note: Hierarchy jump symbols are generated only for critical path and cone schematic fragments. These symbols are shown and explained in the screen shots accompanying the critical path analysis and cones for fanin or fanout logic.

Schematic Fragments for Critical Path Analysis Example

The schematic viewer displays a critical path in schematic form. This allows you to concentrate only on the critical path objects. You can view the whole critical path even if the path traverses multiple levels of hierarchy. Refer to Screen 8-4 and use these steps to create a similar critical path:

1. Start with a design that includes technology information. When the synthesis is complete you can view the critical path in one of the following two ways:
   View Critical Path Method 1: You can click on the View the Critical Path icon on the toolbar to bring up the critical path for viewing, then continue to step 2.
   View Critical Path Method 2: Click on Report flow tab then click on Report Delay power tab. On Report Delay click to select ■ *Bring Up Schematic* and click on Report Delay button. The critical path schematic comes up for viewing. Continue to step 2.

   Note: ❑ *Bring up Schematic* is off by default.
2. The critical path schematic in the viewer window provides delay information for instances, ports, and nets.

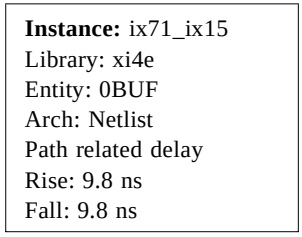
3. Next, RMB over the schematic to popup a menu. On the popup menu select ■ *Object Query* mode.

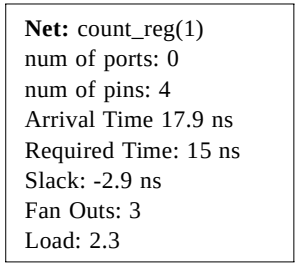
4. Now you can point at any of the schematic object(s) on Screen 8-4 and view context-sensitive popups that are similar to the following:

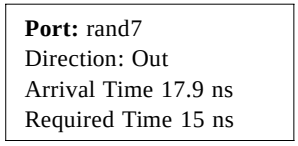
Note: The context-sensitive popup information for the critical path contains timing information. This information is in addition to the standard query popups.

Note: Ensure that the appropriate schematic viewer tool bar filters are enabled. These filters are enabled by default.

Examples of context-sensitive popups for critical path instances, nets, and ports:





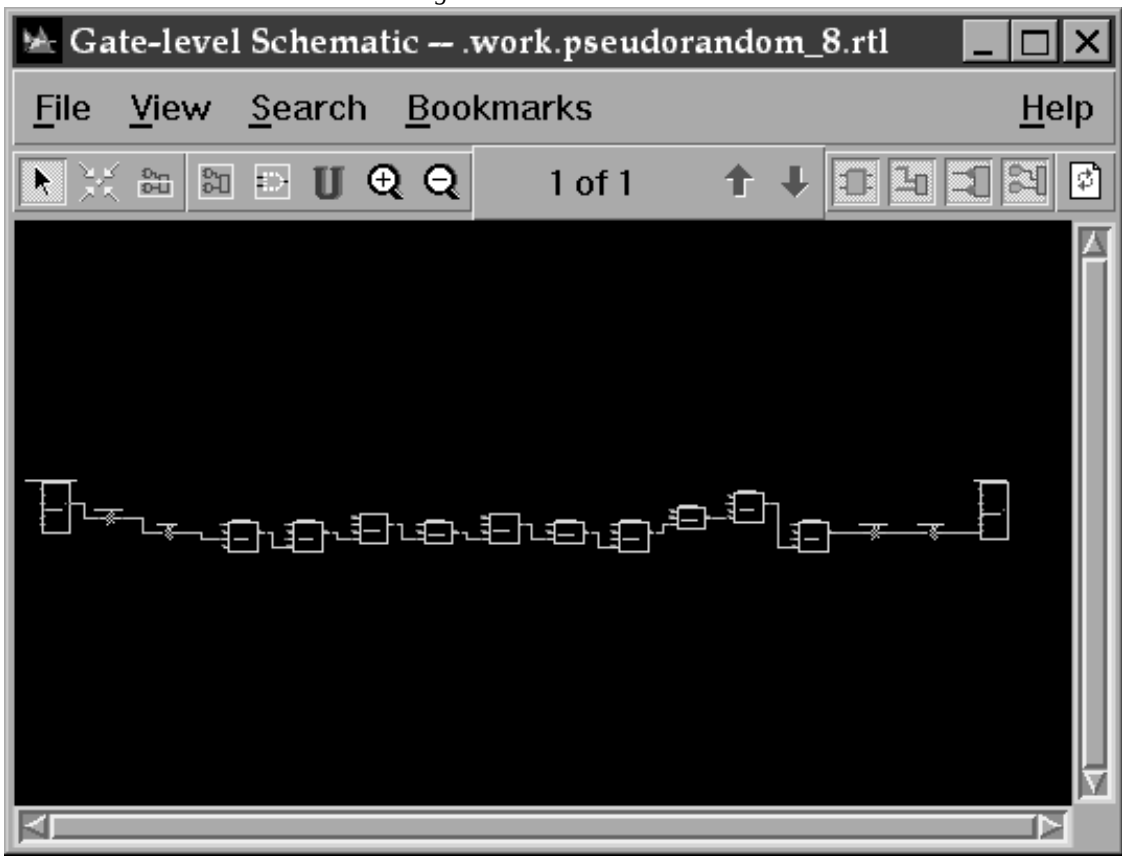


5. The information window shows a critical path (report_delay) report. The graphical critical path schematic in the view window relates directly to the critical path report. The report shows only net names while the schematic view shows names for instances, ports, and nets.

Hierarchy Jump Symbols in the Critical Path

On the far left and far right on Screen 8-4, side-by-side hierarchy jump symbols are shown in the critical path. These symbols indicate that a signal has changed one level up or down. A signal can only change one level at a time. Two or more side-by-side hierarchy jump symbols may occur, as required. Refer ahead to Screen 8-8 for example of zoomed in jump symbols.

Screen 8-4. Critical Path Schematic Fragment



Schematic Cones for Fanin or Fanout Logic from Selected Nodes

Schematic cones are available on every valid schematic. Cones allow you to inspect a fanout or a fanin signal path so you can determine if the logic created by LeonardoSpectrum is acceptable. The fanin cone shows how the selected object is driven, while the fanout cone shows what the selected object is driving. Cones allow you to concentrate on specific parts of the design which may cross hierarchy boundaries. Refer to Screens 8-5 and 8-6. Use these steps:

1. Bring up the schematic viewer. For example, on the LeonardoSpectrum menu bar click on Tools then click on View RTL Level or Gate Level. The RTL Level was used in this example.

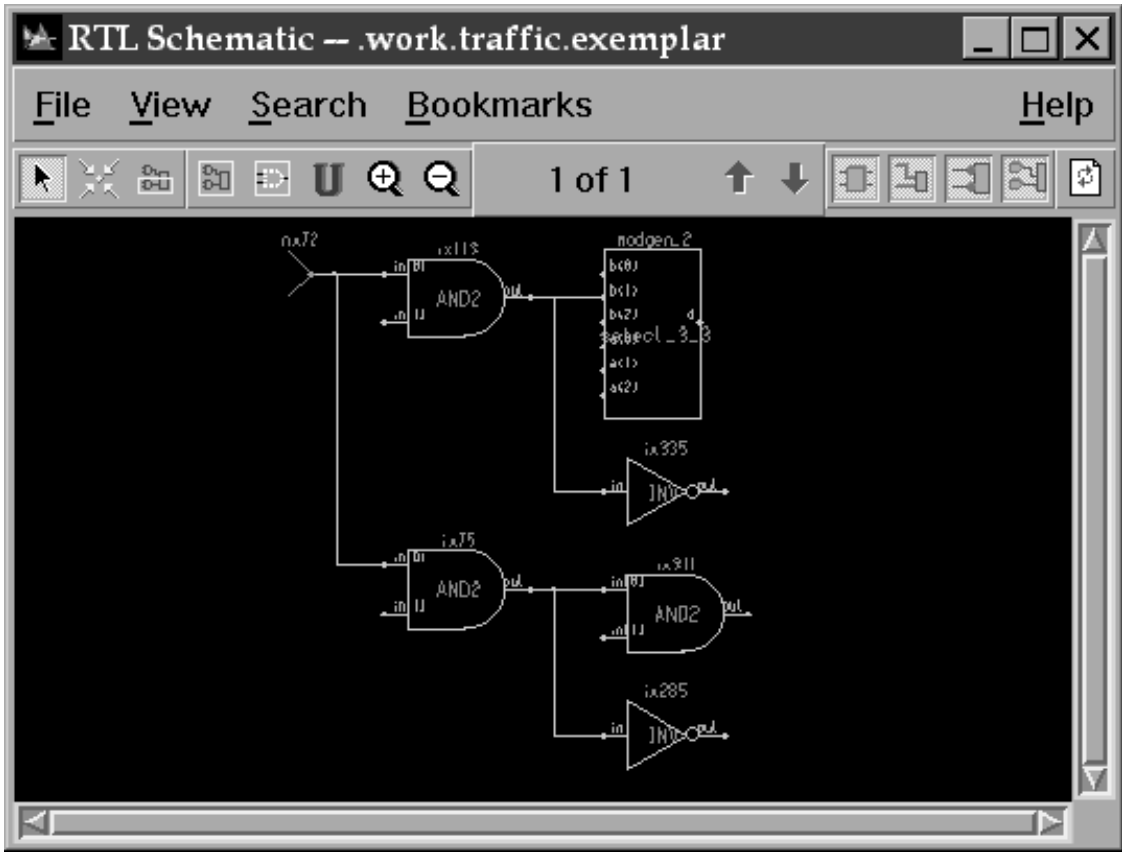
2. On the schematic, select a single net or instance to be the origin of the cone. On Screens 8-5 and 8-6, a net (nx72) was selected.

3. RMB over the schematic to open the popup for tracing cones forward (fanout) or backward (fanin) for a certain number of levels. Click on Trace Forward, 2 Levels. Screen 8-5 clearly shows 2 levels of logic.

4. The > symbol shown at the left on Screen 8-5 shows that the net tracing started from object nx72 (node).

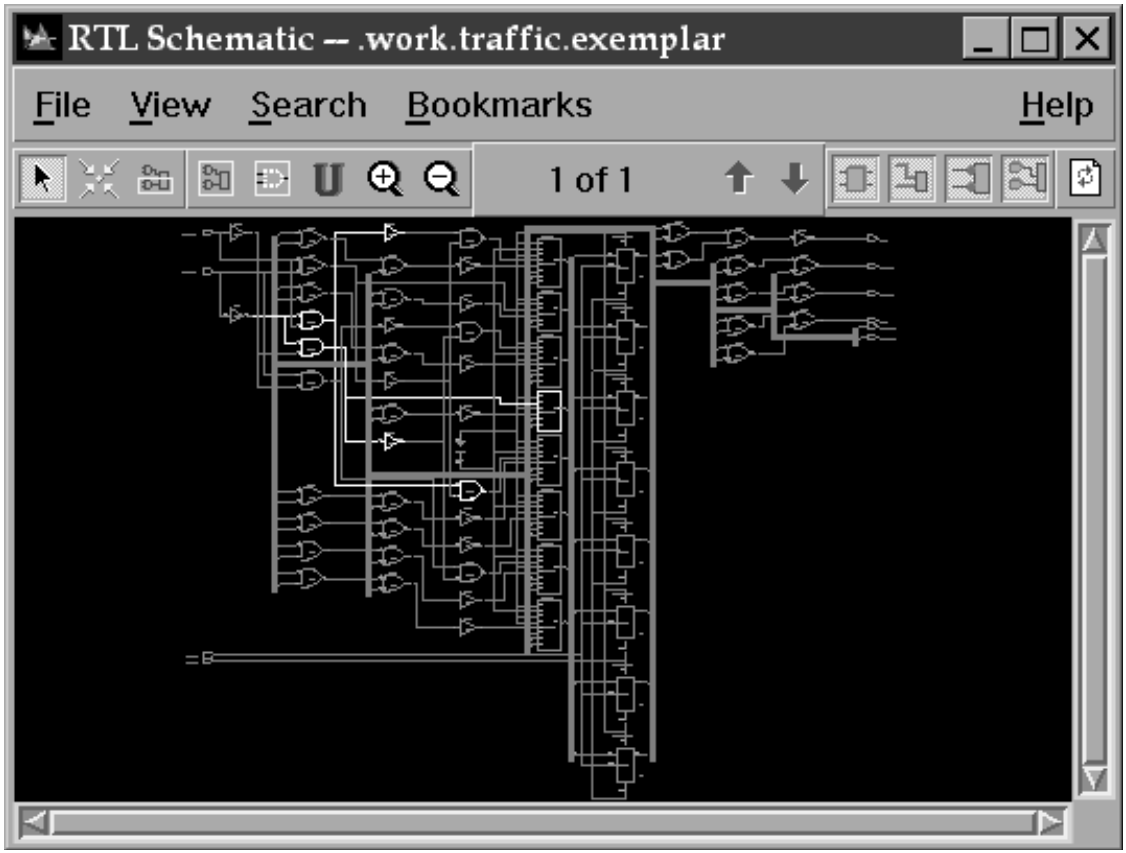
5. Refer to Screen 8-6. Screen 8-6 shows the whole design from which the cone was separated. All of the objects in the cone schematic have been selected; these objects are also selected in the schematic of the whole design. This comparison allows the schematic cone to be viewed in the context of the whole design.

Screen 8-5. Zoomed Cone Fragment Example



Note: The 2 levels of cone logic shown in Screen 8-5 were separated from the whole design as shown in Screen 8-6.

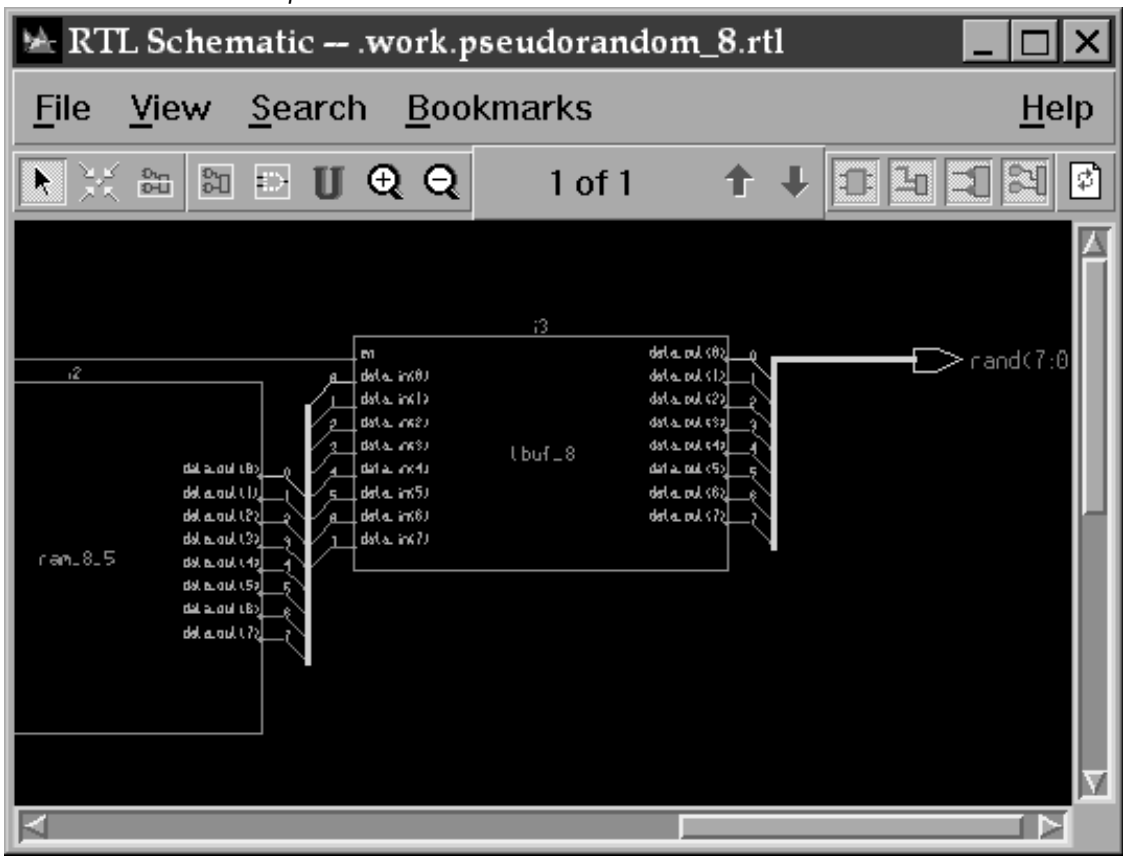
Screen 8-6. Cone Fanout Logic Example in the Whole Design



Example of Cone Tracing with Symbol for Hierarchy Jumps

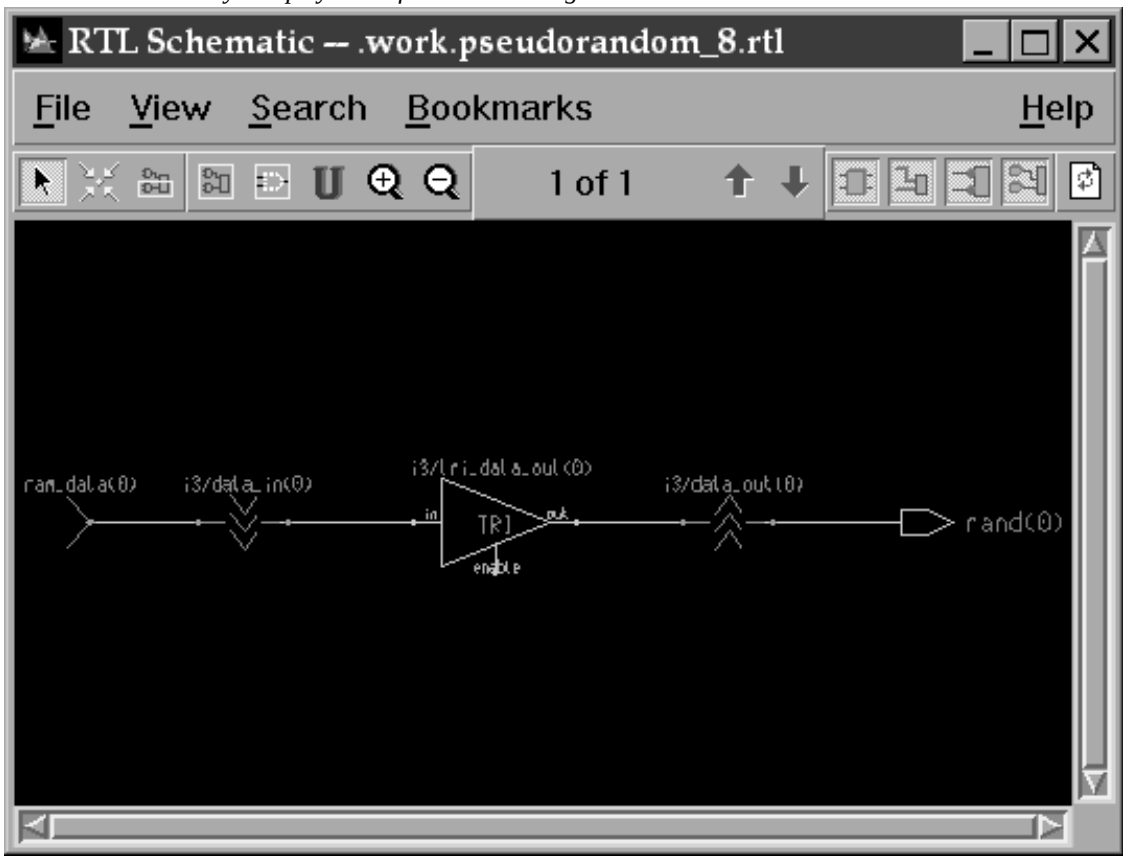
Refer to Screens 8-7 and 8-8. During critical path analysis or when tracing signal logic for cones, you may see the symbol(s) for hierarchy jump(s) as shown on Screen 8-8. The symbol is created by LeonardoSpectrum and indicates that the signal is changing levels of hierarchy. Figure 8-8 is a logic cone trace starting at net ram_data(0). The signal goes down into instance i3, passes through a tristate, and then goes back up out of instance i3. Instance i3 is shown in a normal schematic view in Figure 8-7.

Screen 8-7. Schematic View for Instance i3



Note: In addition to illustrating instance i3, Screen 8-7 also identifies an output bus connection (in a distinct width of gold) with 0-7 port connections (yellow). The colors, gold and yellow, are arbitrary.

Screen 8-8. Hierarchy Jump Symbols for Cone Tracing



Note: The tristate device shown on Screen 8-8 is connected to input pin 0 (data_in(0) and to output pin 0 (data_out (0) on the i3 instance shown on Screen 8-7.

Schematic Viewer Description

LeonardoSpectrum creates and displays a schematic for a selected instance in the current netlist of the design. As shown in Screen 8-9, the name of the schematic is displayed at the top: **work.pseudorandom_8.rtl**. This schematic includes all nets and instances of the selected instance(s). If an instance is not selected, then the selected view appears.

The schematic viewer is intended to be a visual directory for viewing the objects in your design. The menu bar and tool bar above the displayed schematic are organized to assist you with navigating through the design. The group and ungroup commands are available for **Level 3** applications as needed.

Note: Refer to Chapter 12 for setting controls to partition a schematic into sheets.

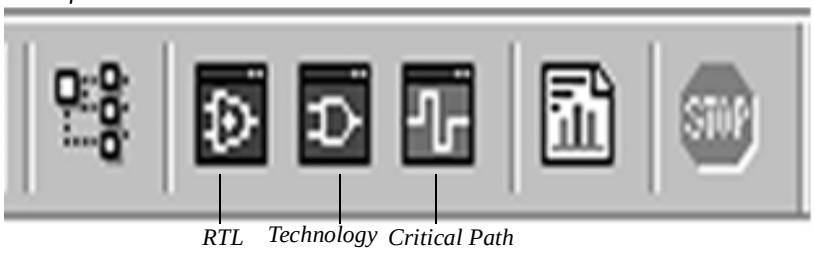
Bring Up Schematic Viewer

Note: the schematic viewer is available after you synthesize a design.

Note: You cannot type the view schematic command at the MSDOS prompt, or in the UNIX shell. The schematic viewer must be launched from the LeonardoSpectrum GUI.

There are several ways to invoke the schematic viewer directly or indirectly. All the methods create a dynamic correspondence between a valid schematic and the current netlist in memory.

Part of Menu Bar/Tool Bar with Schematic Icons



- Click on the appropriate tool bar schematic icon.
- Click on Tools on the menu bar to view the RTL or gate level design.
- Click on view schematic from the HDLInventor to display the objects generated by your source code.
- Click on view schematic from the design browser to view the object selected in the design browser.

- Click on ❑ *Bring up schematic* on report delay; a critical path view comes up. This selection is off by default.
- Click, if necessary, on ■ *Bring up schematic* on the back annotate flow report; a critical path view comes up. This selection is on by default. Read in the design; loading a library and optimization is not required.
- You can type the view_schematic command in the LeonardoSpectrum GUI interactive shell in the Information Window - **Level 3.**

How to Use the Schematic Viewer

This section is divided as follows:

- Menu Bar
- Tool Bar
- RMB Popup Menu
- Strokes

Screen 8-9. Schematic View Header

Menu Bar

The following items are on the menu bar: File, View, Search, Bookmarks, and Help.

File

Print	Print the contents of the current window. Click on Print to bring up the Windows print utility.

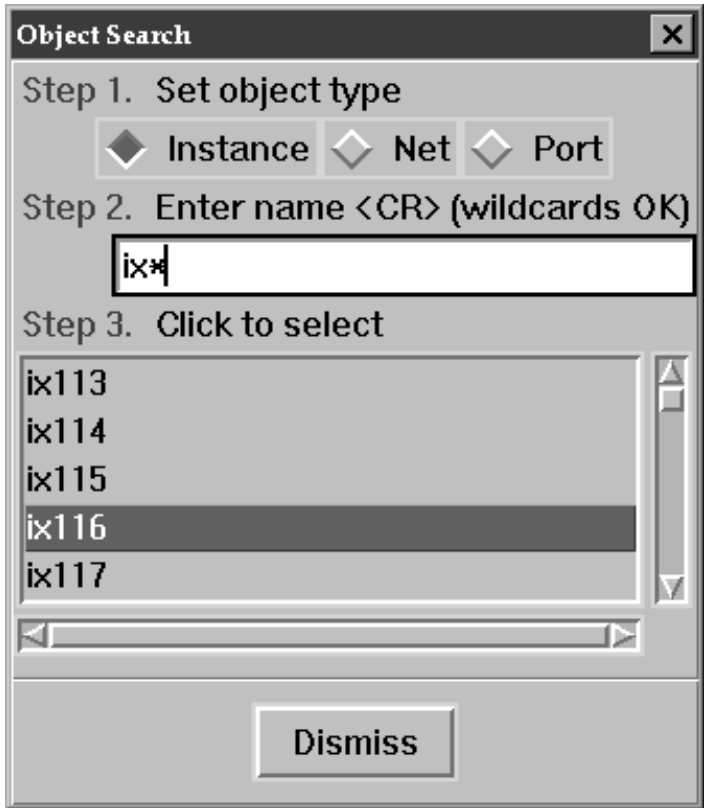
View

View All (**a**) (also see Strokes)	Fits the entire sheet in the display area.
View Selected	Zoom in to view objects that are selected. The view is magnified or reduced to allow viewing of all selected objects, whether close together or far apart. Click again to remove selection from object(s).
Selection Mode:	Set the LMB (left mouse button) mode.
Select Object	Click with LMB to select an object. Hold Shift and click with LMB to add a selected object. Click with LMB and drag to select multiple objects.
Center View	Centers the view around the location of a LMB click, even if you click on empty space.
View Area	Click and drag a boundary around the object(s) with LMB to be viewed.
Zoom **I**n (**I**) (also see Strokes)	Magnifies the field of view of all objects - selected and not selected.
Zoom **O**ut (**O**) (also see Strokes)	Reduces the field of view.

Search

Object Search	You can search on a valid or invalid schematic. Refer to Screen 8-10.
Step 1: Set Object Type	Instance: Click to enable Instance radio button. Net: Click to enable net radio button. Port: Click to enable port radio button.
Step 2: Enter Name <CR>	Type in instance, net, or port name: ix94 (for example), or use wildcards: ix*. Hit **Carriage Return** to perform the search process.
Step 3: Click to select	The matches made on the name in Step 2 are listed in this window. Click on a listed object(s) to highlight and select. The schematic viewer zooms in to display the object selected. If your design has more than one sheet, then the viewer pops up the correct sheet. If the selected object(s) are on more than one sheet, then the viewer arbitrarily views a sheet. Use Windows Shift, Ctrl and drag rules to highlight and select object(s) from the list for viewing.

Screen 8-10. Search



Bookmarks

Bookmarks allow you to mark individual sheets. You can then quickly switch between these “marked” sheets.

Add Current Sheet	Click to add current sheet to bookmark list.
Delete>	Click delete to delete a bookmark.
Change	Click on Change to change (edit) bookmark name.
Bookmarks	This is a list of your bookmarks.
Design Root (select 2 of 6)	This **example** bookmark is attached to the pulldown menu.

Help

❑ Show Tooltips	Tooltips are available when you point the cursor at the tool bar icons.
❑ Increase Help	Click to increase level of detail for tooltips.

Tool Bar

The selection, operation, and filter icons are on the toolbar. These are mutually exclusive icons - once selected - stay selected

Selection Mode	Click with LMB to select an object. Hold Shift and click with LMB to add a selected object. Click with LMB and drag to select multiple objects.
Centered Mode	Centers the view around the location of a LMB click, even if you click on empty space.
View Area Mode	Click and drag a boundary around the object(s) with LMB to be viewed.

Operation Icons - Select Once

View All	Fits the entire sheet in the display area.
View Selected	Zooms in to view objects that are selected. The view is magnified or reduced to allow viewing of all selected objects, whether close together or far apart. Click again to unselect object(s).
Unselect All (**U**)	Click to unselect all selected object(s).
Zoom In	Magnifies the field of view of all objects - selected and not selected.
Zoom Out	Reduces the field of view of all objects - selected and not selected.
1 of x Current Sheet ID Identifies the Current Sheet Number	Click to bring up sheet 1 to x. You can select the sheet you want to view. The controls for partitioning the schematic into sheets are defined under Options on the LeonardoSpectrum menu bar. (Refer to Chapter 12.)
Open Up Arrow Highlighted when enabled.	Click on open up to open a schematic that is one hierarchical level above selected instance.
Open Down Arrow Highlighted when enabled.	Click on open down to open a schematic that is one hierarchical level below selected instance.

Filter Icons

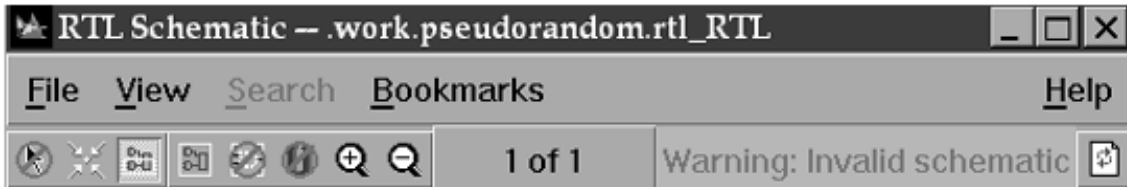
Filter Buttons - Highlight to Enable One or All (default)

Instance Filter	Masks or unmasks instances. Filtering affects selecting, viewing, or pointing to popup context-sensitive information.
Net Filter	Masks or unmasks nets. Filtering affects selecting, viewing, or pointing to popup context-sensitive information.
Pin Filter	Masks or unmasks pins. Filtering affects selecting, viewing, or pointing to popup context-sensitive information.
Port Filter	Masks or unmasks ports. Filtering affects selecting, viewing, or pointing to popup context-sensitive information.

Update Schematic

Update Schematic	Reestablish correspondence between database and schematic. After the schematic becomes valid again, the view is set to the sheet being viewed before updating, if available. Otherwise, the view is set to Sheet 1. Refer to Screen 8-11.

Screen 8-11. Schematic View - Invalid



RMB Popup Menu

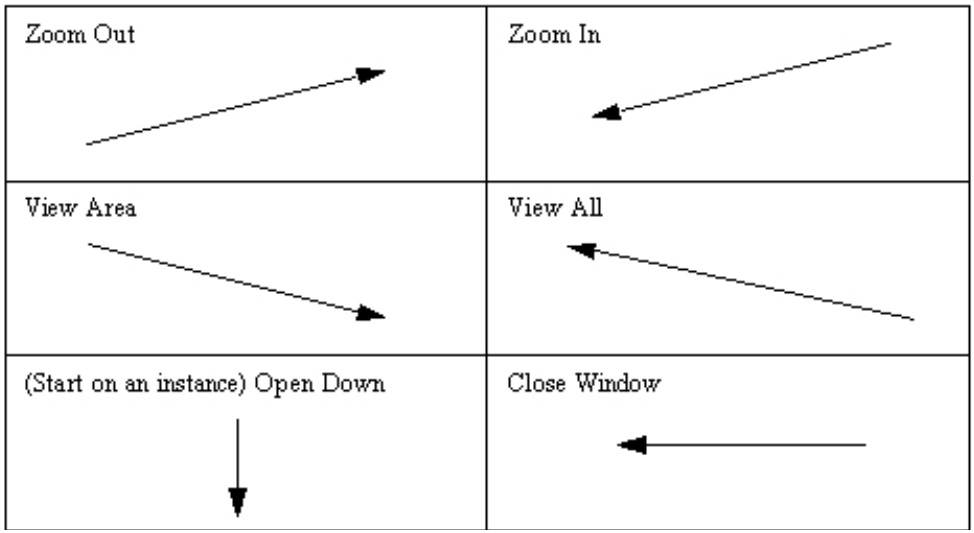
RMB over a valid schematic to popup this menu.

■*Object Query* **Q**	Click on ■ *Object Query* or hit **Q** to enable. Point at objects to popup context-sensitive information.
Add bookmark **B**	Click on Add bookmark or hit **B** to enable. Bookmarks appear on the pull down menu under Bookmarks.
Trace Forward>	1 level, 2, 3, 5 to outputs - trace forward to get depth of cone of fanout logic
Trace Backward>	1 level, 2, 3, 5 to inputs - trace backward to get depth of cone of fanin logic.
Group (**Level 3**)	Click on Group. The selected instances become a new level of hierarchy.
Ungroup (**Level 3**)	Click on Ungroup to dissolve hierarchy of selected object(s).
Ungroup All (**L3**)	Click to issue the command `ungroup -all -hier`.
View>	All (**a**): Fits the entire sheet in the display area.
View>	Selected: Zooms in to view objects that are selected. The view is magnified or reduced to allow viewing of all selected objects, whether close together or far apart. Click again to unselect object(s).
Selection Mode>	Select Object: Click with LMB to select an object. Hold Shift and click with LMB to add a selected object. Click with LMB and drag to select multiple objects.
Selection Mode>	View Centered: Centers the view around the location of a LMB click, even if you click on empty space.
Selection Mode>	View Area: Click and drag a boundary around the object(s) with LMB to be viewed.

Zoom> (**I**)	**I**n: Magnifies the field of view of all objects - selected and not selected.
Zoom> (**O**)	**O**ut: Reduces the field of view of all objects - selected and not selected.
Net: nx540	grayed out

Strokes

Strokes involves holding CTRL and the left mouse or middle mouse button and dragging the cursor in the direction of the arrow shown in the illustration. A red line guides you. Strokes are mutually exclusive. Strokes opens up your schematic with pushing and popping. Strokes is another way to invoke the menu bar, tool bar, and popup menu commands shown in the illustration. **Note**: Open Up is available in the opposite direction of Open Down.



Design Browser Description

The Design Browser is a graphical presentation of the design database. Objects selected and highlighted in the design browser are also highlighted in the schematic viewer. Moreover, if the selected object initiates cross probing, then that line of code is highlighted in the HDL source code. **Note**: Refer to Chapter 12 for Design Browser View Options. These options allow you to customize the design browser objects.

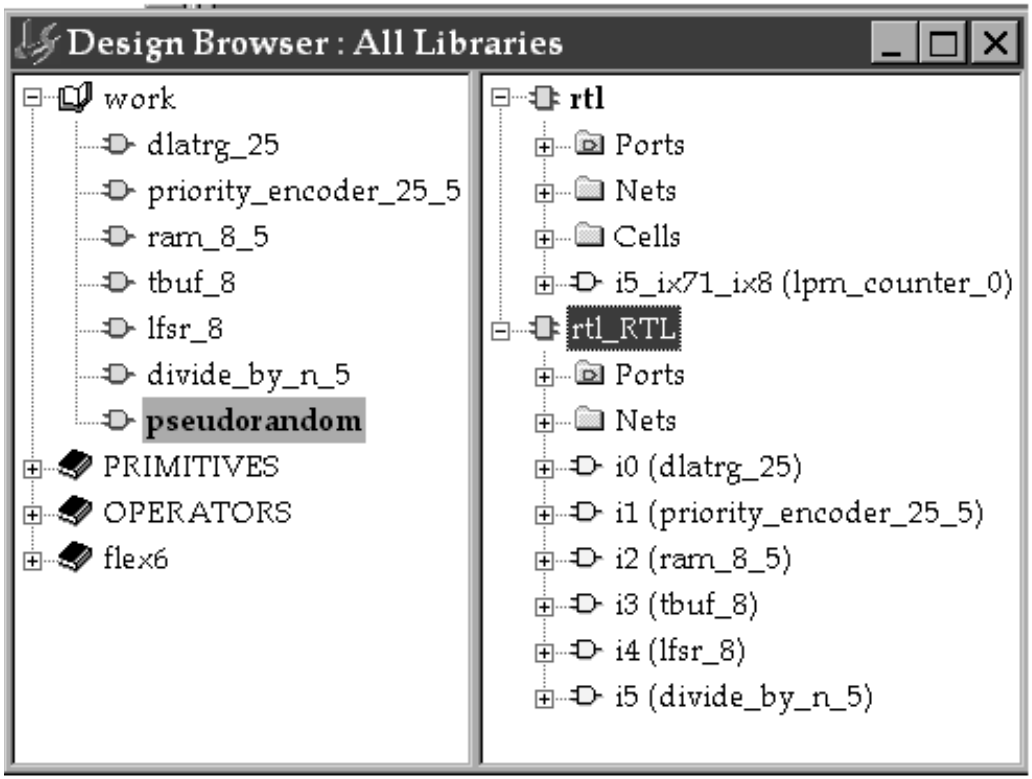
Interactive, filtered views of the design browser are embedded on certain tabs for setting constraint and optimization information on specific objects.

Click on the tool bar icon to open the design browser. The design browser shows the current netlist and your original design.

Browser Windows

The design browser is presented in two windows: library on the left and hierarchy on the right. You can expand and collapse the tree in both windows. Refer to Screen 8-12.

Screen 8-12. Example of Design Browser Windows

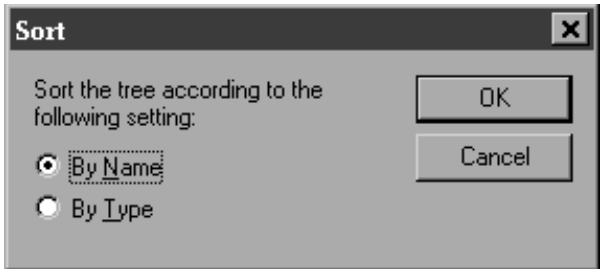


Library Window

The library window contains the libraries, cells, and views that correspond to the netlist in the design database. If you RMB over any of the library items, **Sort...** opens then a click opens the Sort window, Screen 8-13.

Property ALT+Enter	currently unavailable
Sort	You can choose to sort by object name or object type.

Screen 8-13. Sort



Refer again to Screen 8-12. Screen 8-12 shows the following items in the Library window:

work	When your design is read in, LeonardoSpectrum inserts primitives and stores the design in a default library called work.
PRIMITIVES	These are the library cells that LeonardoSpectrum uses to build your design. For example, generic AND2, XOR, OR.
OPERATORS	LeonardoSpectrum uses operators in your design, if your source code includes operators. For example, multipliers, counters, adders are operators.
flex6	This is the example Altera FLEX 6K library. After optimization your FLEX 6K design should contain only instances of cells from the FLEX 6K library.

Hierarchy Window

The hierarchy window displays information about a particular part of the design. You can manipulate your design in this window with the present design, unmap, unfold, and dont_touch commands. Select the commands on the popup menu or refer to the Command Reference for actual syntax. Refer again to Screen 8-12 which shows two example designs for pseudorandom: rtl (optimized) and rtl_RTL (original copy). Refer to optimization in Chapters 3, 4 and 7 for more information.

Hierarchy Window Popup Menus

A popup menu is available in hierarchy window. Click to highlight netlist item, for example, (rtl_RTL). RMB over the highlighted item to popup this menu:

Open Schematic Viewer	Bring up the schematic viewer showing the design that is selected in the design browser.
Trace to HDL Source (Refer to Chapter 2.)	Click with RMB to bring up the HDL source code display. Code line(s) that initiate cross probing are highlighted, if any.
Set as Present Design	This is the same as command `present_design` loaded for your design, LeonardoSpectrum needs to know which file to open for you. Click on Set as Present Design to identify your file. LeonardoSpectrum can now perform design-related operations on this design. When you read in a design from a design file, the present design is set to the top-level view.
Unmap (Refer to `decompose_luts`, Commands chapter, Command Reference)	This is the command `unmap`. The `optimize` command maps PRIMITIVES to your example target, Altera FLEX 6K design. The `unmap` command unmaps your target, Altera FLEX 6K design back to PRIMITIVES. After unmap the design may be different structurally, but is the same functionally as before optimization.
Unfold	This is the command `unfold`. Design instances are folded by default. The folded state allows LeonardoSpectrum to represent multiple hierarchical instances of the same cell with a single view. For example, if you have three instances for the same cell in your design, then all three instances refer to this one cell. When you issue the `unfold` command, the cell is copied and each of the three instances refer to one of the copies of the cell. This allows you to customize the cells as required.
Noopt	This is the attribute `noopt`. Specifies that an instance should not be optimized. **Note**: In contrast to `dont_touch`, lower level hierarchy and leaf instances are not protected from optimization.
Dont Touch (Refer to `auto_write` in the Utilities chapter, Command Reference)	This is the attribute `dont_touch`. Objects marked with `dont_touch` are not optimized or unmapped. Select the desired object(s) and click on Dont Touch. In contrast to `noopt`, `dont_touch` prevents optimization of the lower levels of hierarchy and leaf instances.

Hierarchy Object Popup Menu

RMB over objects in the hierarchy window for the following nets, ports, and cells popups:

Nets and Ports	
Cells	
Group*	Click on Group. The selected instances become a new level of hierarchy.
Ungroup*	Click on Ungroup to dissolve hierarchy of selected object(s).
Ungroup All Levels*	Click to issue the `ungroup -all -hier` command. Dissolves hierarchy from the selected block down to all blocks lying below the selected block.
Unmap	This is the command `unmap`. The `optimize` command maps PRIMITIVES to your example Altera FLEX 6K design. The `unmap` command unmaps your Altera FLEX 6K design back to PRIMITIVES. After unmap the design may be different structurally, but is the same functionally as before optimization.
Noopt	This is the attribute `noopt`. Specifies that an instance should not be optimized. **Note**: In contrast to `dont_touch`, lower level hierarchy and leaf instances are not protected from optimization.
Dont Touch (Refer to auto_write, Utilities chapter, Command Reference)	This is the attribute `dont_touch`. Objects marked with `dont_touch` are not optimized or unmapped. In contrast to `noopt`, `dont_touch` prevents optimization of the lower hierarchy levels and leaf instances.
Trace to HDL Source (Refer to Chapter 2.)	Click with RMB to bring up the HDL source code display. Code line(s) that initiate cross probing are highlighted, if any.

* Commands are gray for LeonardoSpectrum Level 2.

Note: Two new design browser features:

(1) lib.cell.view reference for design browser. Objects are displayed in the design browser using a lib.cell.view nomenclature. Libraries are displayed in the left column of the design browser. Each library may contain one or more cells and each cell may contain one or more views. The view is the corresponding netlist associated with the cell.

(2) busses are indexed (collapsed).

Adding a Technology Specific Symbol Library

Exemplar Logic provides symbol libraries for most technologies.

If your technology does not have a symbol library, then install the required symbol library in the `$EXEMPLAR/data/symlibs` directory as follows:

```
copy <library>.sglib $EXEMPLAR/data/symlibs
```

Note: The library must be in the `<library>.sglib` format.

Design Database for Level 3

9

LeonardoSpectrum turns your HDL code into a design database while LeonardoInsight provides tools for exploring and interacting with the design. This chapter provides a brief tour of the design database and describes methods for using commands on the interactive command line shell.

Design Data Information Model

All design data is stored in a set of libraries which start at the root. A library contains a list of cells, and a cell contains a list of views. For most designs, a cell has only a single view. Views are the basic building blocks of your design. A view is the implementation or contents of a single level of hierarchy.

Examples:

- When you read a VHDL description into LeonardoSpectrum, your VHDL entity translates to a cell, and the VHDL architecture (contents) translates to a view. By default, the cell is stored in a library called `work`.
- When you load a technology library into LeonardoSpectrum, it becomes a library in the design database, which contains all of the cells of that technology.
- LeonardoSpectrum creates a library of PRIMITIVES automatically. This library represents all binary functions that LeonardoSpectrum may require when compiling or elaborating HDL (VHDL and Verilog) descriptions.
- LeonardoSpectrum also automatically creates an OPERATORS library. This library contains operator cells (adders, multipliers, muxes). When compiling HDL descriptions, these operators are generated when needed.

The following objects are contained by a view and are used to represent netlists and hierarchies in a design:

- A view has ports, nets and instances.
- A port is a terminal of a view.
- An instance is a pointer to a view.
- A net is a connection between ports and/or port instances (pointer to the port of the view under an instance).

This code example is a small VHDL description which represents a binary AND function:

```
entity and2 is
   port (a,b: bit; o:out bit);
end and2;
architecture contents of and2 is
begin
   o <= a AND b;
end contents;
```

LeonardoSpectrum then creates a cell called `and2` in the default library `work`. The cell contains a view, called `contents`. The view contains three ports: `a`, `b` and `o`. The view also contains an instance of a view in the LeonardoSpectrum PRIMITIVES library. This is an instance of a primitive AND. The name of the instance is created by LeonardoSpectrum. The view also contains three nets: `a`, `b`, and `o`, connecting the instance to the ports of the view. All objects libraries, cells, views, ports, nets and instances can contain attributes.

Accessing Design Data

The Design Browser allows you to graphically browse through all libraries and the design hierarchy. If you are using the LeonardoSpectrum interactive command line shell to write scripts, then use the following guidelines to identify objects in the database. To identify an object in the design database, LeonardoSpectrum uses a formalized design naming convention. Any object in the database is accessible from a single root (the set of libraries). The root is identified by the design name `.` (dot). A library is identified by the design name:

```
.library_name
```

The general design name for a view is:

```
.library_name.cell_name.view_name
```

Wildcards and regular expressions are accepted and expanded in design names to identify multiple objects simultaneously.

LeonardoSpectrum also has a present design. This is a design name that identifies the top of your design hierarchy. When LeonardoSpectrum starts up, the default is set to the root (.). After you read in a design file, the present design is set to the top level view as described in the file. Table 9-1 is a list of commands that enable you to investigate any object in the design database using absolute design names or relative design names.

Table 9-1. List of Commands

Command	Description
`present_design`	Displays the present design name.
`present_design <design_name>`	Changes the present design to <`design_name`>.
`list_design -ports`	List all ports in the present design.
`list_design -nets`	Lists all nets in the present design.
`list_design -instances`	Lists all instances in the present design.
`list_design <design_name>`	Lists all objects contained in <`design_name`>.
`list_attributes`	Lists all attributes in the present design.
`list_attributes -port <port_name>`	Lists all attributes on the port <`port_name`> of the present design.
`push_design <design_name>`	Changes the present design to <`design_name`>. This precompiled Tcl procedure is defined in `exemplar.ini` file and allows you to change the present design while returning to your starting point.
`pop_design`	Returns you to the present design before the last `push_design`. This precompiled Tcl procedure is defined in `exemplar.ini` file and allows you to change the present design while returning to your starting point.

Table 9-2 shows examples for the commands listed in Table 9-1.

Table 9-2. Example of Commands

Command	Description
`list_design .work`	Lists all cells in the library called `work`.
`list_design -ports .work.and2.contents`	Lists all ports on the view `contents` of the cell `and2` in the `work` library.
`list_design .*`	Lists all cells in all libraries.
`list_design -nets`	Lists all the nets in the present design (only valid if present design is a view).
`list_design -instances x`	Lists the design name of the view under the `instance x` of the present design.
`present_design .work.and2.contents`	Changes the present design to view `contents` of cell `and2` in the `work` library.
`push_design inst_1`	Changes the present design to the view to which the instance `inst_1` is pointing.
`pop_design`	Changes the present design back to before the previous `push_design` call.

Note – The formalized naming convention can uniquely identify libraries, cells and views in a single name. However, since a view can contain three different types of objects (ports, nets, instances), there is a problem identifying these uniquely. For example, the name:

```
.l.c.v.x
```

does not identify an object `x` in view `v` of cell `c` in library `l` as a port, net or instance. To work around this problem, the `list_design` command (and other commands that accept nets, ports or instances) all have an option (`-port` , `-net`, or `-instance`) to identify an object type.

The result of `list_design` is a (Tcl) list, which can easily be used in scripts. The following example script reports how many cells are in each library in the database:

```
for each i [list_design .] {
   set the_length [llength [list_design $i]]
   puts “library $i contains $the_length cells”
}
```

After the demo `mancala.vhd` file, for example (`$EXEMPLAR/LeoSpec/demo`), has been read and the `act2` library loaded, this script will produce the following output:

```
library .PRIMITIVES contains 19 cells
library .work contains 2 cells
library .OPERATORS contain 6 cells
library .act2 contains 925 cells
```

The object separator is programmable; the default is `.` (dot). You can change the separator by setting the Tcl variable `list_design_object_separator`. For example, the following script prints the present design name, changes the object separator, and prints the design name again:

```
puts “The present design is [present_design]”
set list_design_object_separator /
puts “The present design is [present_design]”
```

Produces the output:

```
The present design is .work.mancala_32.exemplar
Info: setting list_design_object_separator to /
The present design is /work/mancala_32/exemplar
```

LeonardoSpectrum notifies you with a message when it recognizes the setting of a LeonardoSpectrum built-in variable, rather than a normal Tcl variable.

Use the following example commands to list commands and variables in the interactive command line shell:

`help` (lists all commands)
`help present_design` (lists options and information)
`help list*` (lists all list commands)
`help -variables` (lists all variables)
`help -var write*` (lists information about write variables)

Refer also to the LeonardoSpectrum Command Reference guide.

Application Notes 10

This chapter contains application notes for:

- Pipelined Multiplier
- APEX 20K/20KE PTERM Support
- Save and Restore Project

Pipelined Multiplier

This pipelined multiplier feature is currently implemented by LeonardoSpectrum for: Actel a54sx, Altera FPGA, ORCA, and Xilinx FPGA.

Introduction

Pipelining a combinational logic involves putting levels of registers in the logic to introduce parallelism and, as a result, improve speed.

Flip flops introduced by pipelining typically incur a minimum of additional area on FPGAs, by occupying the unused flip flops within logic cells that are already used for implementing combinational logic in the design.

The LeonardoSpectrum Approach to Pipelining

LeonardoSpectrum requires certain constructs in the input RTL source code description to allow the pipelined multiplier feature to take effect. These constructs call for "m" levels of registers to be inferred at the output of the multiplier,

where m is an integer greater than 1.

Let n be the smallest integer that is greater than or equal to the base 2 logarithm of the width of the multiplier/multiplicand. LeonardoSpectrum automatically pipelines the multiplier by moving the first x levels of the inferred registers into the multiplier, where

```
x = m-1, for 2 <= m <= n
```

or

```
x = n-1, for m > n
```

Variable

The pipelined multiplier feature is turned on by default. This feature can be disabled by setting the variable `pipeline_mult` to false.

```
set pipeline_mult false
```

Quality of Results

Post layout areas and delays are presented for Altera FLEX 10K and Xilinx Virtex. A comparison is made between the non-pipelined version and the 4-stage pipelined version of a 16-bit unsigned multiplier; the non-pipelined version has one level of register at the output.

As shown for these technologies, the speed improvements are significant. Moreover, in the context of an entire design, the percentage of additional area is minimal.

Altera FLEX 10K	P & R Area (LCs)	Delay, ns	Mhz
non-pipelined	541	27.4	36.49
pipelined	587	12.8	78.12

Xilinx Virtex	P & R Area (slices)	Delay, ns
non-pipelined	156	23.353
pipelined	192	11.268

Note: *Clock Enable, Asynchronous Clear/Set*

In the final pipelined multiplier, common clock enable and asynchronous clear are supported in all levels of registers.

Asynchronous set is supported with these restrictions: Among the levels of registers inferred at the output of a multiplier, only the last level can have asynchronous set. If asynchronous set is present on an inferred register level other than the last, then the multiplier is not pipelined; a message appears to that effect.

Example Template VHDL and Verilog RTL

The following is an RTL level description of a 16-bit, unsigned multiplier with the operand inputs registered. LeonardoSpectrum infers four levels of registers at the output of the multiplier and generates a 4-stage pipelined multiplier.

- VHDL Template
- Verilog Template

VHDL Template RTL description for an unsigned pipelined multiplier:

```
library ieee ;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

entity pipelined_multiplier is

    -- generic size is the width of multiplier/multiplicand;
    -- generic level is the intended number of stages of the
    -- pipelined multiplier;
    -- generic level is typically the smallest integer greater
    -- than or equal to base 2 logarithm of size, as returned by
    -- function log, which you define.

    generic (size : integer := 16; level : integer := log(size));
    port (
```

```
      a : in std_logic_vector (size-1 downto 0) ;
      b : in std_logic_vector (size-1 downto 0) ;
      clk : in std_logic;
      pdt : out std_logic_vector (2*size-1 downto 0));

end pipelined_multiplier ;

architecture exemplar of pipelined_multiplier is


    type levels_of_registers is array (level-1 downto 0) of
    unsigned (2*size-1 downto 0);
    signal a_int, b_int : unsigned (size-1 downto 0);
    signal pdt_int : levels_of_registers;

begin

    pdt <= std_logic_vector (pdt_int (level-1));

    process(clk)
    begin
    if clk'event and clk = '1' then

       -- multiplier operand inputs are registered

       a_int <= unsigned (a);
       b_int <= unsigned (b);

       -- 'level' levels of registers to be inferred at the
       -- output of the multiplier

       pdt_int(0) <= a_int * b_int;
       for i in 1 to level-1 loop

          pdt_int (i) <= pdt_int (i-1);

       end loop;

    end if;
    end process;

end exemplar ;
```

Verilog Template RTL description for an unsigned pipelined multiplier:

```
module pipelined_multiplier ( a, b, clk, pdt);

/*

 * parameter 'size' is the width of multiplier/multiplicand;
```

```
 * parameter 'level' is the intended number of stages of the
 * pipelined multiplier;
 * which is typically the smallest integer greater than or equal
 * to base 2 logarithm of 'size'
 */
parameter size = 16, level = 4;
input [size-1 : 0] a;
input [size-1 : 0] b;
input clk;
output [2*size-1 : 0] pdt;

reg [size-1 : 0] a_int, b_int;
reg [2*size-1 : 0] pdt_int [level-1 : 0];
integer i;

assign pdt = pdt_int [level-1];

always @ (posedge clk)
begin
    // registering input of the multiplier
    a_int <= a;
    b_int <= b;

    // 'level' levels of registers to be inferred at the output
    // of the multiplier
    pdt_int[0] <= a_int * b_int;
    for (i = 1; i < level; i = i + 1)
    pdt_int [i] <= pdt_int [i-1];
end

endmodule
```

Altera APEX 20K/20KE PTERM (Product-Term) Support:

The "implement in pterm" option in LeonardoSpectrum allows the module for the selected instance to be implemented as sum-of-products. The appropriate project assignments are then passed to the Quartus compiler to map the module into Embedded System Block (ESB) PTERM WYSIWYG elements.

Currently, LeonardoSpectrum does not map to PTERM WYSIWYG elements. When the "implement in pterm" option is selected, the block to be mapped to PTERM is flattened by default.

Usage:

non_GUI (tcl command line):

```
implement_in_pterm <instance_name or block name>
```

For example:

```
implement_in_pterm inst_a
```

```
implement_in_pterm.work.cell_a.rtl
```

LeonardoSpectrum GUI, Constraints Flow Tab, Module editor: Select a block and click on the "implement in pterm" button. Apply changes.

Save and Restore Project

LeonardoSpectrum is introducing the powerful concept of *projects*. A project systematically organizes your design and the implementation of the design. A project allows you to checkpoint your design and then restore the project at a later time. When you save and restore a project, redundant runs are avoided and productivity is increased. This chapter is divided as follows:

- What is a Project
- Advantages of Using Project
- Starting a Project
- Tips and Tricks - Examples

What is a Project?

A project consists of three files:

1. **Design Database .xdb file**: The .xdb file is your design representation. This file is in a non-ASCII format. The .xdb file can only be read by LeonardoSpectrum. The XDB format is Exemplar Logic's proprietary format.

2. **GUI settings .lsp file:** The .lsp file stores the GUI setting of your project. This file is an ASCII file. The .lsp file contains information like input file path, output file path, FlowTabs options settings, and window settings. **WARNING**: Avoid editing this file. Editing this file may change the look and feel of the GUI and also the synthesis run of your design.

3. **Project Settings .scr file:** The .scr file contains several variables which affect the flow of synthesis.

Advantages of Using Project

There are three main advantages:

1. **Power of Check Pointing for Level 3:** You can store the design and the design implementation while you are working on a project. For example, you have optimized your design and then decide to quit the tool. If you save the optimized design as a project before you quit, then the optimization is not lost. Later, when you restore the project, the optimized design is waiting for you to continue the task with further timing optimization or generating reports. This checkpoint process proves to be very useful and time saving with large designs.

2. **Organizes a design and the synthesis runs in a systematic way:** For example, you want to try your design implementation on two different devices (parts) for your Altera FLEX 6K technology. Device 1 is EPF6016QC208 and Device 2 is EPF6016QC240. In addition, you want to analyze the trade-off between report area and report delay for each of the Altera FLEX 6K devices. Now you try your design on Device 1 and Device 2 and then save each as a project:
   - Device 1 - save as project: `high_speed.lsp` file (.lsp is the project file extension for LeonardoSpectrum).
   - Device 2 - save as project: `low_cost.lsp` file.

   Later, during your team presentation, for example, you can quickly open the `high_speed.lsp` and `low_cost.lsp` projects. You then click the report tab on the FlowTabs to review the reports for area and delay for the two Altera FLEX 6K devices.

3. **Portable: take your project with you from platform to platform**. For example, if you are working on a module on a PC, and would like to continue the task on a workstation, then you can! **Caution:** If you read in your input files, then you must ensure that the files installed at the workstation are in the same directory structure as at the PC.

Caution: Within the v1999.x series, your saved project files are forward and backward compatible, and you can read v1998.x project files in v1999.x. However, your project files are not backward compatible between the v1999.x series and v1998.x series of LeonardoSpectrum. The FSM encoding "auto" default in v1999.x series causes the project incompatibility since the v1998.x series does not have the "auto" FSM encoding default. If you want to read a v1999.x project in v1998.x then set the following variable in 1999.x to v1998.x:

```
xdb_write_version v1998.x
```

Starting a Project

Click File -> for the project related commands. Refer to Chapter 2 and use these steps:

1. Start a new project and run the flow for this task.

2. Next save this project as: `my_new_project.lsp`

   Note Level 3: You can save a project at any checkpoint during synthesis (for example, after completing optimization), and then open the project and resume the task at a later time.

3. File -> Recent Projects> shows your saved projects. Select your project, `my_new_project.lsp` from this list; LeonardoSpectrum then loads the project.

Level 3 Interactive Command Line Shell

The following commands are available for managing your project in the interactive command line shell.

```
save_project_script <project_name>
```

```
restore_project_script <project_name>
```

The option [`-no_design`] is available for *do not save design* or *do not restore design*.

You can open projects completed in the GUI on the interactive command line shell.

Tips and Tricks - Examples:

Tip 1

Use these steps for applying a large design to one or more technologies, for example: Altera FLEX 6K and Xilinx 4000E.

1. Read in your design for Altera FLEX 6K, for example, and save the project as: `my_design_elab.lsp`
2. Next, continue with your implementation of the project: `my_design_elab.lsp`
3. Now if you want to experiment with `my_design_elab.lsp` on another Altera FLEX 6K device or on the Xilinx 4000E technology, then you merely load `my_design_elab.lsp` again and start the task. You do not need to read the input files again.
4. You optimize your design and save as `my_optimize_design.lsp` project.

5. Then you load this project and make several timing optimization runs. You may experiment with timing constraints until the needs of your design are met. During this trial and error process you only run timing optimization; re-reading and re-optimizing the design are not necessary. Saving your task as a project proves to be both cost and time efficient.

Tip 2

Take your project with you.

1. Finally, you can move your project from platform to platform by zipping each file:
   - project .lsp
   - project .xdb
   - project .scr
2. Unzip the files in a directory and open the project. Begin your design task!

Tip 3

The following steps are suggestions for saving a fixed set of default settings to be used later as a starting point for any number of projects.

1. Click File->New Project. All settings for LeonardoSpectrum defaults are cleared.
2. Proceed through the FlowTabs for your design and select your settings.
3. Load the target technology library as required for your ASIC or FPGA design.
4. Click File->Save Project As to save the current project to a new filename. For example, `<default_file>.lsp`. Later, you can use the Open Project option to load your saved defaults as needed.
5. When you open the `<default_file>.lsp` for another new project, your default settings are in place. However, you cannot read in a file with this `<default_file>.lsp` file. This file loads a technology library, if a library was included in your default project.

Menu Bar Items

11

The following choices are on the Options and Tools pulldowns:

- Options -> Session Settings...
- Options -> Browser Filter...
- Tools -> Variable Editor

Note: These options are referenced from Chapters 2 and 8.

Note: Click **Cancel** to return to main window. Click **Help** for assistance.

Session Settings

Options -> Session Settings opens with these tabs:

- Editor Options, Screen 11-1 and Table 11-1
- Session Settings, Screen 11-2 and Table 11-2
- Schematic Viewer Properties, Screen 11-3 and Table 11-3

Screen 11-1.Editor Options - Tab (1 of 3)

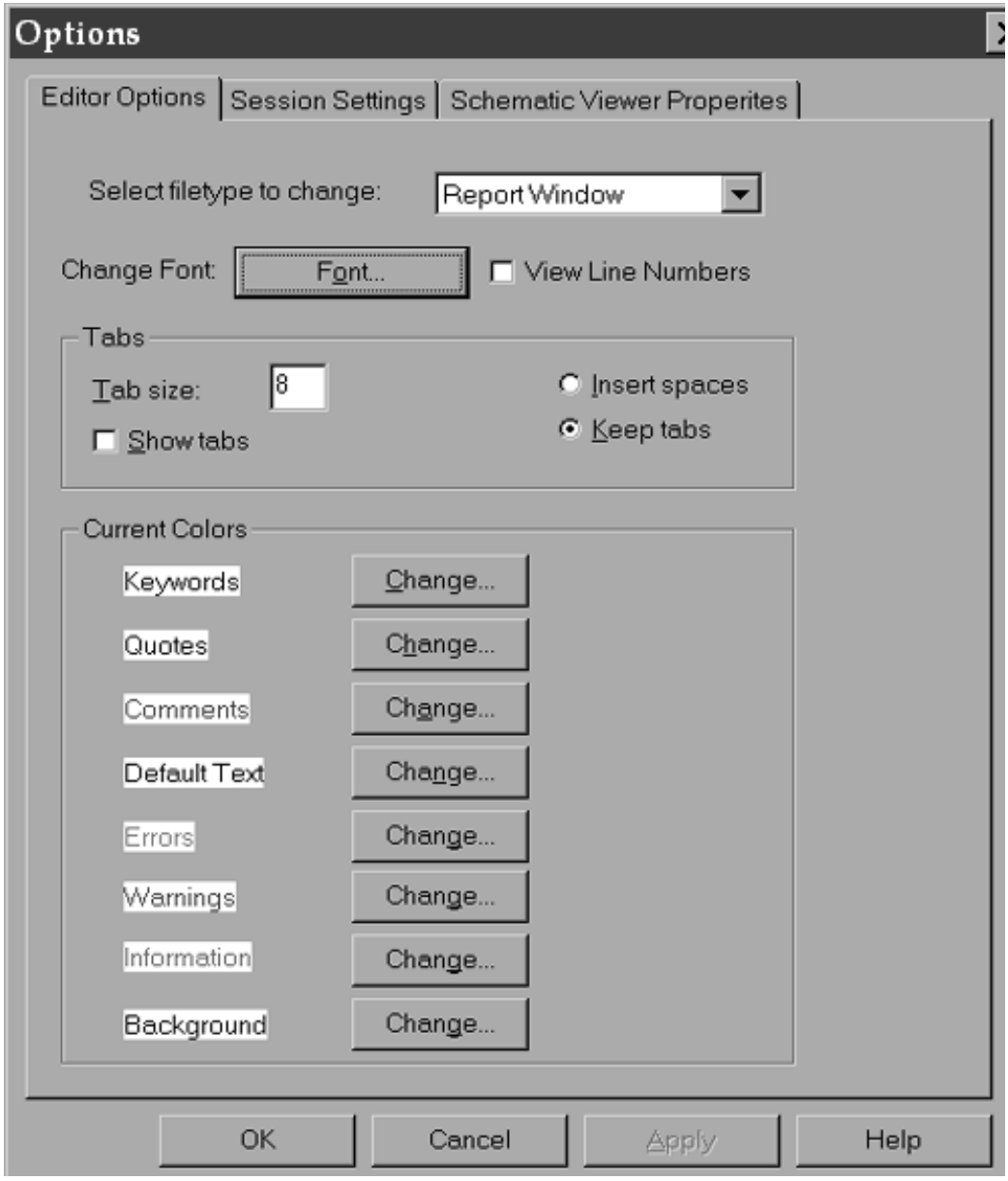


Table 11-1. Editor Options - Tab (1 of 3)

Option	Description	
Select file type to change:	**Scroll** Report Window 🗁	Scroll for: TCL, Verilog, VHDL, XNF
Change Font: **Font Button:** click to open **Font**.	Fonts: Courier, Courier New, Fixedsys, Letter Gothic, Letter Gothic MT, Lucida Console, MS LineDraw	
	Font Sizes:8, 9, 10, 11, 12, 14, 16, 18, 22, 24, 26, 28, 36, 48, 72	
	Font Style: regular, italic, bold, bold italic	
☐View Line Numbers	Check to view line numbers.	
Tabs	Tab size: 8	
	❍Insert spaces	Toggle with Keep tabs radio button.
	☐ Show tabs	Check to show tabs.
	●Keep tabs	Toggle with Insert spaces radio button.
Current Colors with **Change Button** for text foreground. A Color pallet opens. Choose basic colors or add custom colors.	Keywords	
	Quotes	
	Comments	
	Default Text	
	Errors	
	Warnings	
	Information	
	Background	
Click **OK** to apply choices and close. Click **Apply** to apply choices.		

Screen 11-2. Session Settings - Tab (2 of 3)

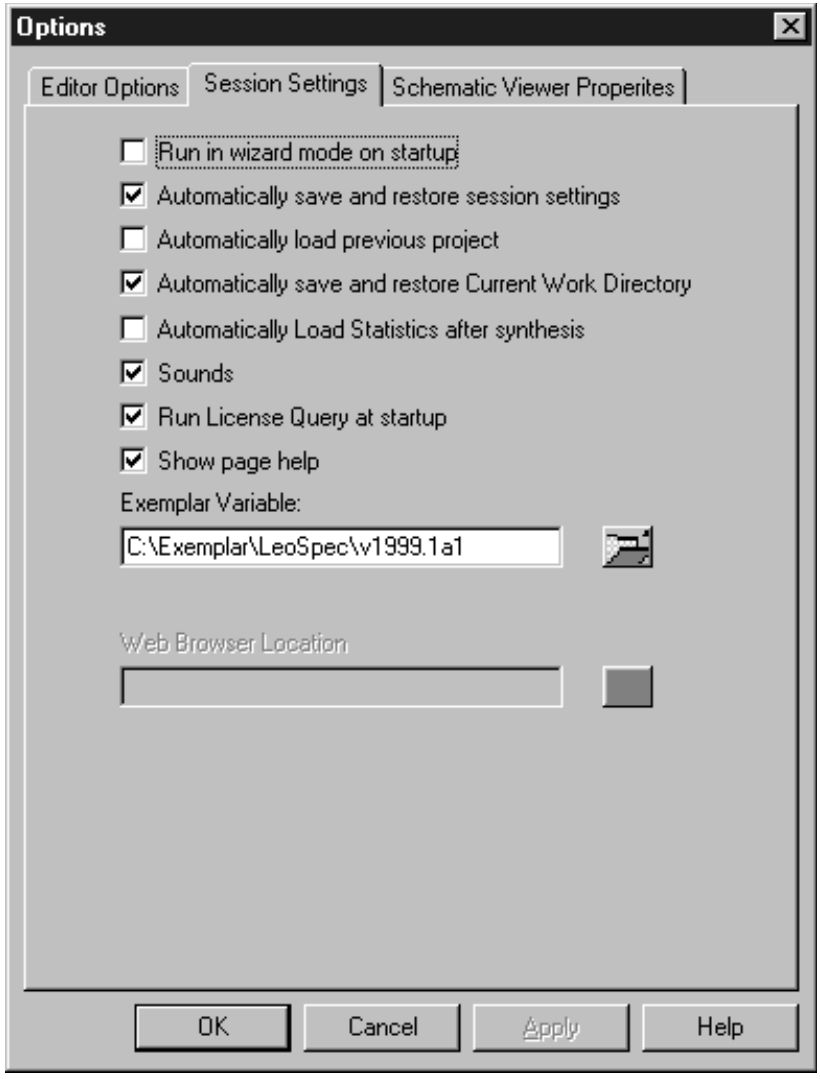


Table 11-2. Session Settings - Tab (2 of 3)

Option	Description
❒ Run Wizard mode on startup:	When you start up, SynthesisWizard does not open automatically unless you select this box. You can also set this option on Input File(s).
■ Automatically save and restore session settings	Settings on all session settings are saved. Your synthesis setup is saved. Default is selected.
❒ Automatically load previous project (off by default)	When this selection is off, LeonardoSpectrum opens without any project loaded. This default is identical to File -> Open Project. When ■ Automatically load previous project is selected, LeonardoSpectrum then opens the previous project, if any.
■ Automatically save and restore Current Working Directory (on by default)	When you start up LeonardoSpectrum, the current working directory is restored. If ❒Automatically save and restore Current Working Directory is off, then a default current working directory is available.
❒ Automatically Load Statistics after synthesis:	The summary of your synthesis run is loaded.
■ Sounds:	Music plays when you start up.
■ Run License Query at startup.	Asks if you want to run with Level 1, 2, or 3.
❒ Run License Query at startup	Disabled if you disable ■Run license selection next time
■ Show page help	Displays text at top of FlowTabs and Power Tab screens.
Exemplar Variable: 🗁	Click to open the set EXEMPLAR variable browser.
Web Browser Location: 🗁	UNIX only: Click to open web browser location. Invokes your web browser when you click on a technology logo.

Click **OK** to apply choices and close. Click **Apply** to apply choices.

Screen 11-3. Schematic Viewer Properties - Tab (3 of 3)

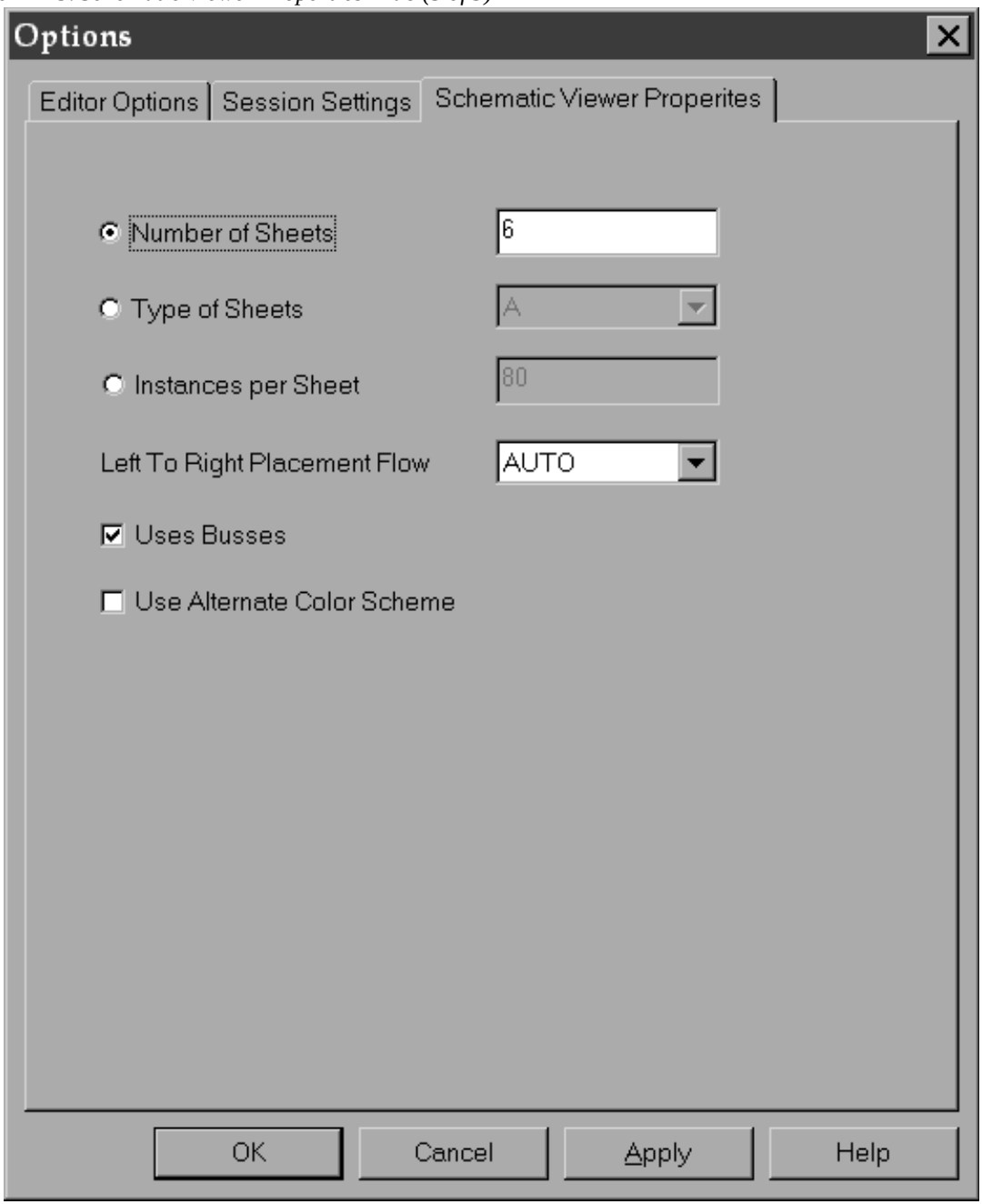


Table 11-3. Schematic Viewer Properties - Tab (3 of 3)

Option	Description
● Number of Sheets	Enter the approximate number of sheets required to divide your schematic for viewing. The schematic viewer partitions your design into sheets based on this radio button choice.
❍ Type of Sheets	Pulldown: A, B, C, D (standard drafting size paper). The schematic viewer partitions your design into sheets based on this radio button choice.
❍ Instance per Sheet	Number of instances on each sheet. The schematic viewer partitions your design into sheets based on this radio button choice.
Left to Right Placement Flow	Pulldown: AUTO
■ Uses Busses	Schematic includes busses.
❒ Use Alternate Color Scheme	You can change viewer colors.

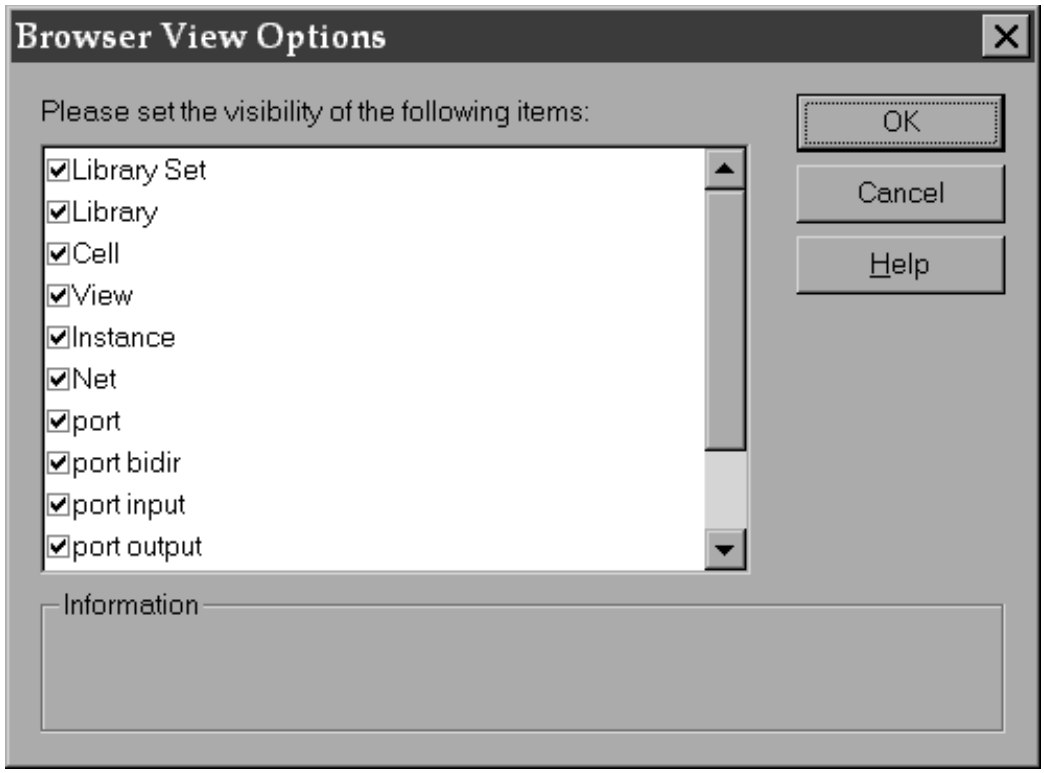
Click **OK** to apply choices and close. Click **Apply** to apply choices.

Note: Refer to Chapter 8, LeonardoInsight

Browser Filter

Options -> Browser Filter opens with Browser View Options, Screen 11-4.

Screen 11-4. Design Browser View Options



The Browser View Options allows you to customize the objects displayed in the design browser. Refer to Chapter 8, LeonardoInsight. Click **OK** to apply and exit.

Variable Editor

Tools -> Variable Editor opens with System Variables, Screen 11-5.

Screen 11-5. Variable Editor

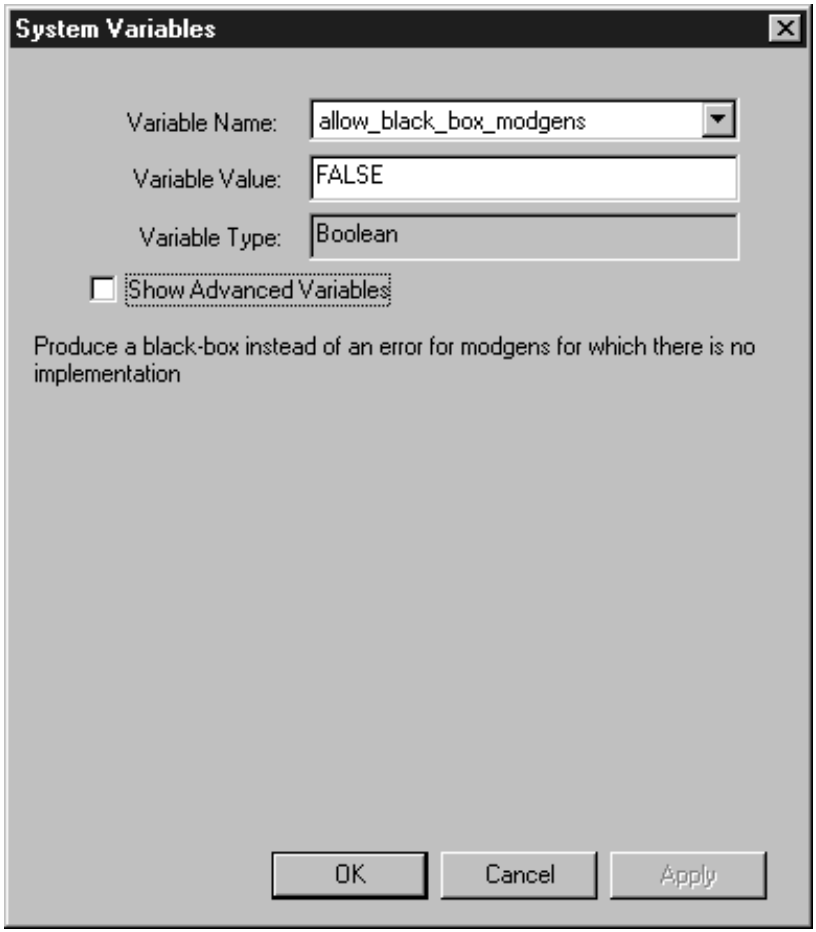


Table 11-4. Variable Editor

Option	Description
Variable Name: Select from pulldown.	These variables are also listed in alphabetical order in the Command Reference Guide.
Variable Value:	Use default FALSE or type in a value.
Variable Type:	Use default Boolean or type in a variable type.
❒ Show Advanced Variables. These are typically uncommon variables.	When ■ Show Advanced Variables is selected, the advanced variables appear in the Variable Name: pulldown. For example: apex_wysiwyg_support is an advanced variable.

Note: Variable editor allows you to select and add variables without typing. Refer to the Command Reference guide, Variables, for a printed list. Advanced variables are not printed in this list.

Click **OK** to apply choices and close. Click **Apply** to apply choices.

Design Methodology

12

This chapter outlines the recommended design methodology when using the LeonardoSpectrum interactive command line shell for ASIC designs with up to 500K gates and for FPGA designs. This chapter is divided as follows:

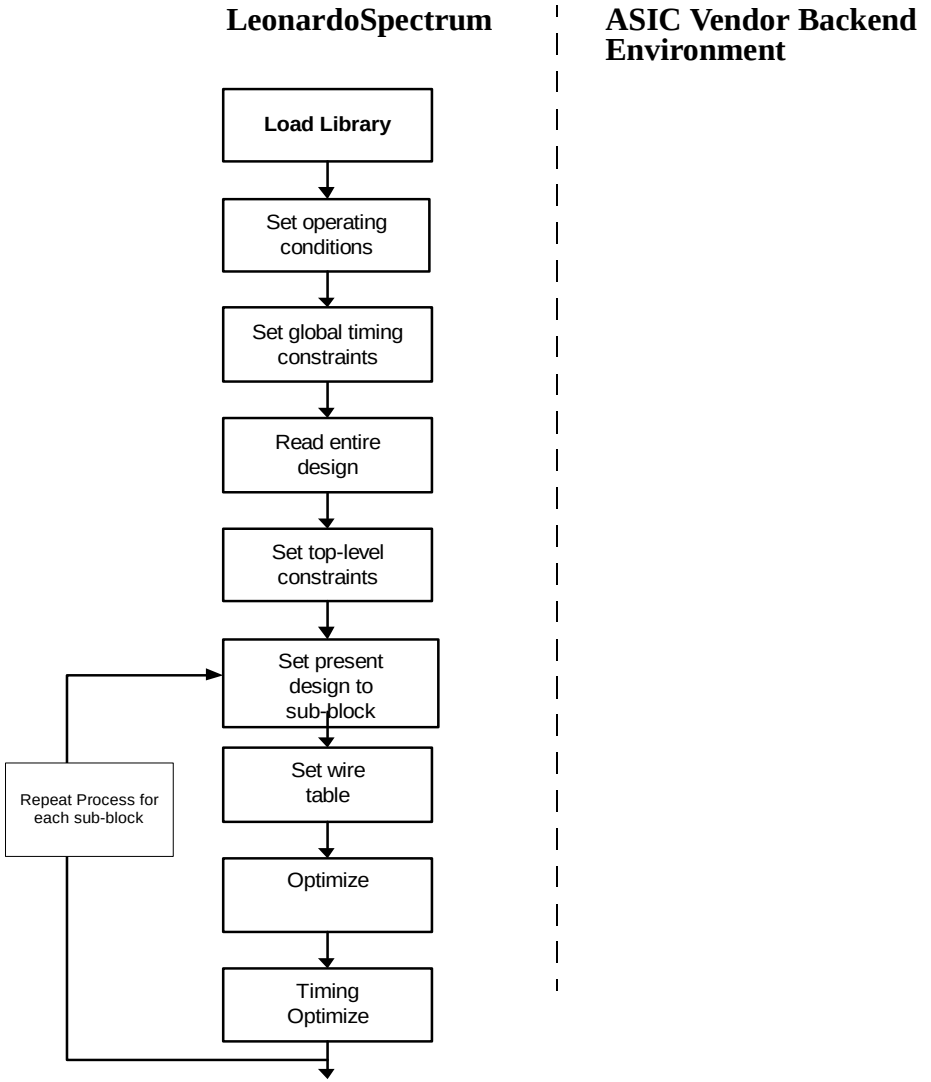
- Flow Charts - **ASIC**
- Example Flow Session - **ASIC**
- Environment Setup - **ASIC**
- Design Partitioning
- Synthesizing Designs
- Adding Cells and Modeling Memories
- Setting Constraints
- Global Optimization Variables - **ASIC**
- Clock Buffering - **ASIC**
- Optimization
- Saving Results
- Back Annotation
- Referencing Netlist Objects
- Hierarchy Manipulation
- Design Compiler Commands - **ASIC**

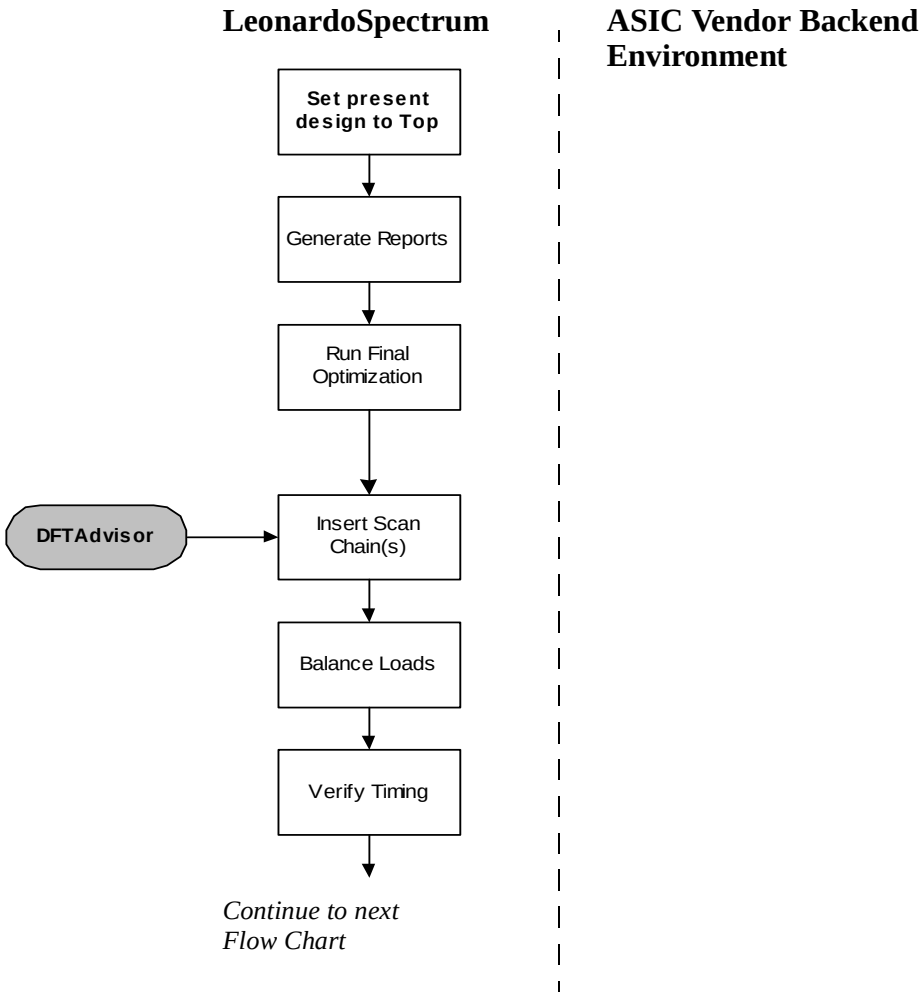
Note: The information in this chapter assumes that you are familiar with the LeonardoSpectrum GUI and documentation. This information is also useful for both ASIC and FPGA environments.

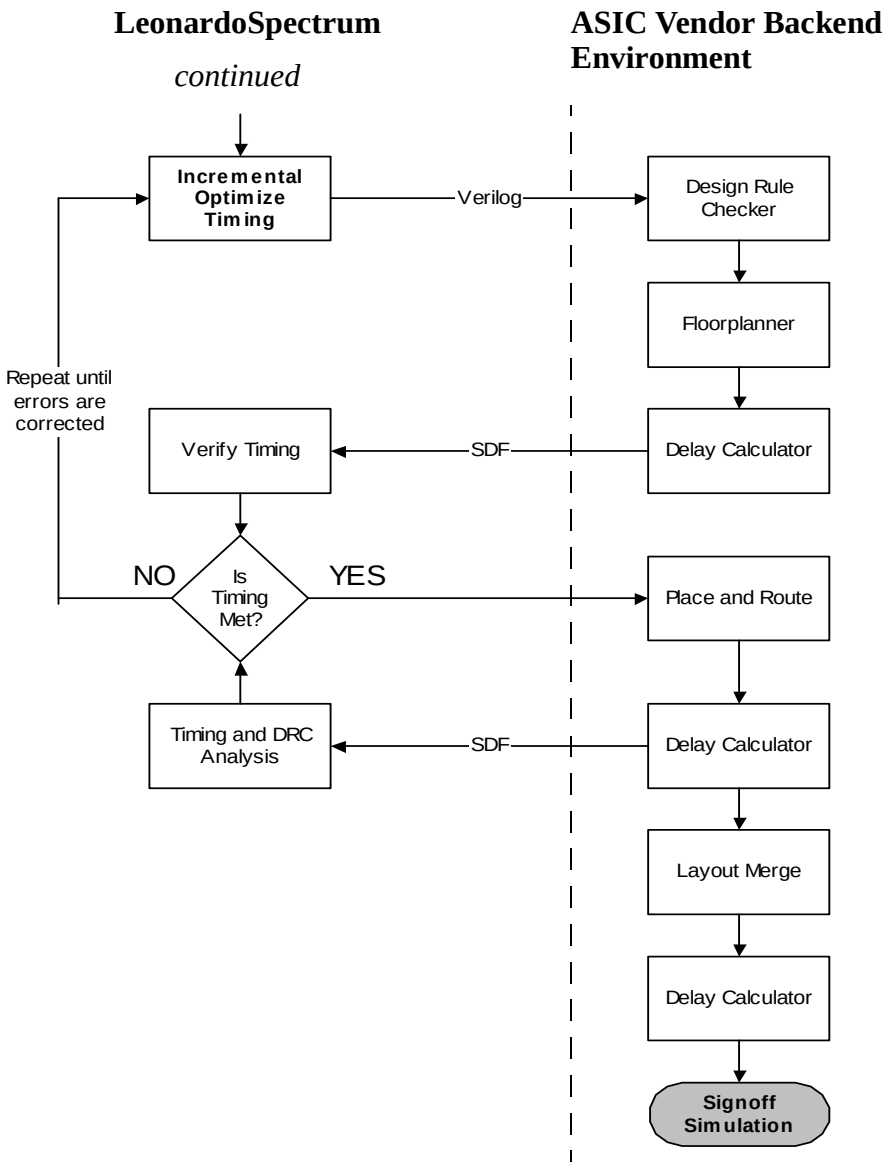
ASIC Flow Charts

The flow charts are presented in three sections:

- Load Library
- Set Present Design to Top
- Verify Timing - Incremental Optimize Timing




LeonardoSpectrum
ASIC Vendor Backend Environment
Set present design to Top
Generate Reports
Run Final Optimization
DFTAdvisor
Insert Scan Chain(s)
Balance Loads
Verify Timing
Continue to next Flow Chart


LeonardoSpectrum
continued
ASIC Vendor Backend Environment
Incremental Optimize Timing
Verilog
Design Rule Checker
Floorplanner
Repeat until errors are corrected
Verify Timing
SDF
Delay Calculator
NO
Is Timing Met?
YES
Place and Route
Timing and DRC Analysis
SDF
Delay Calculator
Layout Merge
Delay Calculator
Signoff Simulation

Example Flow Session

This section contains:

- Flow Steps
- Sample Flow Script

Flow Steps

Commands are issued in LeonardoSpectrum from the command line interface shell. This section provides a detailed example of using LeonardoSpectrum with the Fujitsu place and route tools. This example uses:

- Fujitsu cg61 CMOS library
- Mentor Graphics
  `Inventra lprfir_139 core`.

Note: Each step is explained in more detail in the appropriate section in this chapter.

1. Invoke LeonardoSpectrum
   ```
   % spectrum
   ```
2. Load the Xilinx cg61 library.
   ```
   load_library cg61
   ```
3. Set operating conditions. These have been set to default values in the ASIC library. Refer to the supplied ASIC library documentation for exact numbers.
   ```
   set temp 80
   set process typical
   set voltage 5.0
   set max_fanout_load 16
   set max_cap_load 4
   set max_transition 1.2
   set wire_table worst
   ```
4. Set global timing variables. The timing constraints constrain timing at the top-level. Global constraints define timing for the sub-blocks as the blocks are optimized. Use global timing constraints to constrain sub-block boundary logic to one-half of the clock period (as defined by the register2register variable) to ensure correct timing at the top-level.
   ```
   set input2register 10
   set register2output 10
   ```

5. Read in the HDL files. VHDL design files must be listed in a bottom-up order. Verilog design files can be listed in any order since LeonardoSpectrum automatically detects the top-level module. Be sure to place brackets "{}" around multiple file lists, i.e. `{file1.v file2.v file3.v}`.

Note: LeonardoSpectrum uses file suffixes to identify file formats.

For example, `read lpfir.v`

The following is a partial list; refer to Chapter 2 for a complete list:

- VHDL files: .vhd, .vhdl;
- Verilog files: .v, .ver;
- EDIF files: .edn, .edf, .edif.

6. Set top level timing constraints.

```
clock_cycle 5 clk
arrival_time 2 [all_inputs]
required_time 3 [all_outputs]
```

7. Set present design to first sub-block. Setting the present design swaps a subblock into main memory for optimization. All global constraints previously set apply to the "present design" in memory.

```
present_design block_a
```

8. Set the wire load model to reflect the gate count of the sub-block. The wire load model determines the capacitance value applied to all nets.

```
set wire_table cg61_5000area
```

9. Perform optimization. LeonardoSpectrum performs both area and timing optimization. In this example, you do an optimization to achieve the smallest design. Set the `-macro` switch which disables IO buffer insertion which is deferred to the final, top-level optimization.

```
optimize -ta cg61 -area -macro
```

10. Repeat steps 7, 8, 9, and 10 for each subblock. When done, set present design to the top-level.

11. Generate area and timing reports. The optimization runs display a single area and worst case timing number. Reports are only necessary if more information is required.

```
report_area -cells
```

```
report_delay -num_paths 1 -critical_paths
```

12. Perform final optimization to adjust buffering between sub-blocks and to insert IO buffers. Specific buffers can be set using the “pad” command. Set the “-chip” option for the optimize command to perform IO pad insertion.

```
pad data_in* ITFUH

pad data_out* OBCV

optimize –area –single_level -chip
```

13. Save final netlists. Save the design as an XDB file, which is the LeonardoSpectrum binary data format and as a verilog netlist for backend place and route.

```
write top.xdb

write top.v
```

14. The design is now ready for place and route.

15. Continue with delay calculation, layout merge and signoff simulation for your design.

Sample Script

```
# Loading Target Technology
load_library cg61

# Setting operating conditions
set temp 80
set process typical
set voltage 5.0

# Setting Design Rule Conditions
set max_fanout_load 16
set max_cap_load 4
set max_transition 1.2
set wire_tree worst

# Set global timing constraints
input2register 3
register2output 3

# Define IO Buffers pads
pad data_in* ITFUH
pad data_out* OBCV

# Read complete design
read {block_a.vhd block_b.vhd top.vhd}

# Set timing constraints
clock_cycle 6 clk
arrival_time 3 data_in*
required_time 3 data_out*

# Set present design to subblocks, set wire table and optimze
present_design block_a
set wire_tree cg61_5000area
optimize -ta cg61 -area –macro
present_design block_b
set wire_tree cg61_20000area
optimize –ta cg61 –delay –macro

# Set present design to top level and complete optimization
present_design top
report_area -cells
report_delay -num_paths 1  -critical_paths
optimize –area –single_level –chip

# Save Design
write top.xdb
write top.v
```

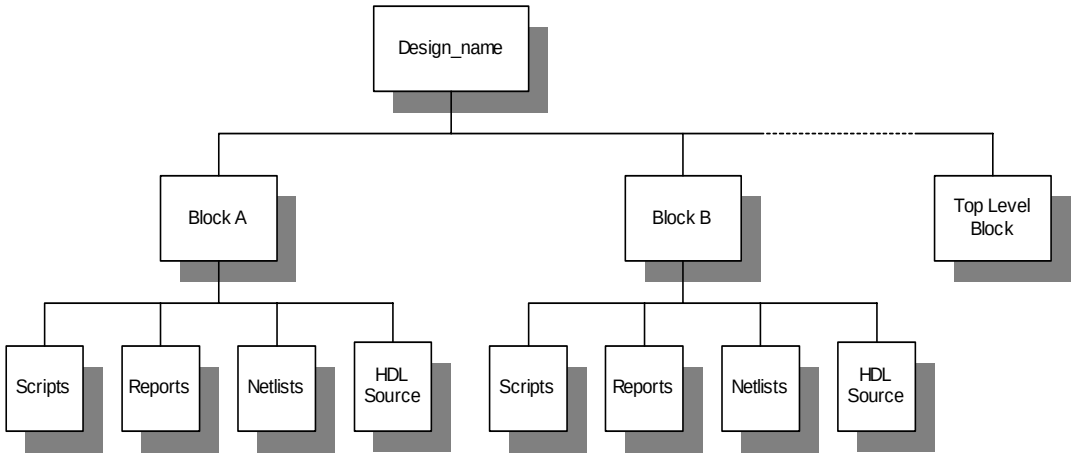
Environment Setup

The directory structure outlines the environment setup.

Directory Structure

Although the choice of a directory structure is often an individual or team decision, Exemplar Logic provides the following recommendation.

Figure 12-1 Directory Structure



- Scripts - Contains all constraint and optimization scripts
- Reports - Contains all area, timing, constraint and environment reports
- Netlists - Contains all optimized, mapped netlists
- HDL Source - Contains the original VHDL or Verilog source code

Startup Files

Startup files can be a useful way to pre-configure LeonardoSpectrum for daily optimizations. Refer to the following example:

`exemplar.ini` Startup File

```
# Define common aliases

alias lp list_design -ports
```

```
alias reportit {report_area; report_delay}

# Set synthesis working directory - Note directory
# slashes are UNIX style for all machines including PCs.

set_working_dir "C:/Exemplar/LeoSpec/v1999.1/demo"

# Disable Asynchronous Feedback Loops

set delay_break_loops TRUE
Startup files for UNIX
```

Place the `exemplar.ini` file in your working directory. The commands in the file are automatically executed when invoking LeonardoSpectrum.

Startup files for Windows

1. Place the `exemplar.ini` file in a personal or project folder that is not part of the Exemplar software install directory structure. All Exemplar software files are deleted and replaced with each new software install.
2. Edit the file `$EXEMPLAR/data/exemplar.ini` directory to add the following line to the bottom of the file. You must add this line again after each new software install.

```
Source d:/<pathname to startup file>/exemplar.ini
```

Setting Aliases

LeonardoSpectrum allows you to set aliases to rename any LeonardoSpectrum command. The `exemplar.ini` startup file is the most logical place to define commonly used aliases. For example:

```
> alias lp list_design -ports
```

Installing ASIC Libraries

The ASIC vendor typically supplies LeonardoSpectrum design kits. The Exemplar Logic web site, `www.exemplar.com/partners` contains the latest library availability and instructions for obtaining each design kit. LeonardoSpectrum design kits are shipped as three files:

- ***Technology_name.syn*** – Binary file containing cell timing, logic and cost data. Place this file in the `$EXEMPLAR/lib` directory
- ***Technology_name.*****vhd** – VHDL file containing operator implementation information. Place this file into the `$EXEMPLAR/data/modgen` directory
- ***Technology_name.*****doc** – Documentation file that provides installation information and route table information. Print this file as a reference.

Adding ASIC Libraries to the GUI

Refer also to Chapter 7 for more library information. Edit the `device.ini` located in the `$EXEMPLAR/lib` directory. This file provides information to the interface needed to display the library in the “Technologies” form pick list. This file must be modified with the following fields:

- **Manufacturer –** Required library manufacturer name.
- **Family –** Required technology family name. Usually is same as library name.
- **Library –** Required library name. Usually is the same as the family name.
- **Symbol Library –** Required symbol file name. Usually is the same name as library name.
- **"FPGA"|"ASIC" –** Required**.** Specifies if library is to be displayed in the FPGA or ASIC pick list. Set to ASIC.
- **SOURCE | DEST | BOTH –** Required, Specifies if a library can be used as a source library when reading in mapped structural netlists, a destination netlist for final optimization or both. Set to “Both”.
- **Vendor Name –** Optional**,** leave blank
- **Contact –** Optional, leave blank
- **html page –** Optional, When a web address is specified in this field then a button appears on the technology FlowTab that invokes a web browser and navigates to that address.

Example `device.ini` file entry

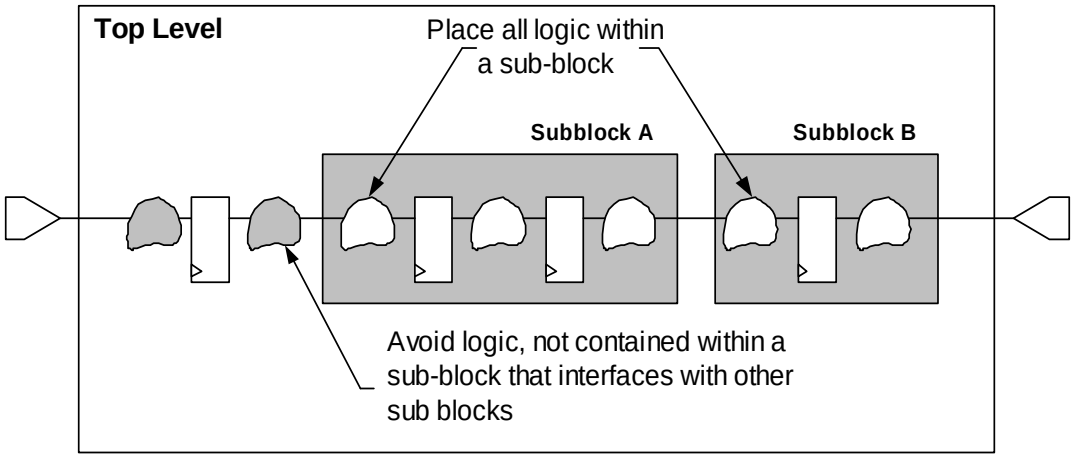
"Fujitsu", "CG61", "cg61", "none", "ASIC", "BOTH", "Exemplar Logic", "" , ""

Design Partitioning

Use the following steps when partitioning a design into leaf blocks:

1. Place all logic into hierarchical blocks; *glue* logic does not exist at any level that is not part of a hierarchical block. Following this convention helps ensure correct results from the LeonardoSpectrum timing analysis environment.

Figure 12-2 Example of Design Partitioning


Top Level
Place all logic within a sub-block
Subblock A
Subblock B
Avoid logic, not contained within a sub-block that interfaces with other sub blocks

2. Gate counts in leaf blocks are between 10K and 50K gates. Optimizations can be performed on blocks much larger provided the sub-hierarchy falls within this guideline.

3. In general, limit clocks to one per block. Multi-clock designs are supported; however, setting constraints becomes more complex.

4. Group similar logic together, i.e., state machines, data path logic, decoder logic, ROMs. Pay close attention to blocks that may contain special area or delay optimizations. For example, if you know a particular block is going to contain the critical path, eliminate any non-critical logic from that block.

5. Place state machines into separate blocks of hierarchy to speed optimization and provide more control over encoding.

6. Separate timing critical blocks from non-timing critical blocks. Note: LeonardoSpectrum performs area and timing optimizations separately. By separating timing critical logic into one block, you can perform aggressive area optimizations on a greater percentage of the design, and create a smaller circuit that meets timing.

Figure 12-3 Example of Separating Timing Blocks

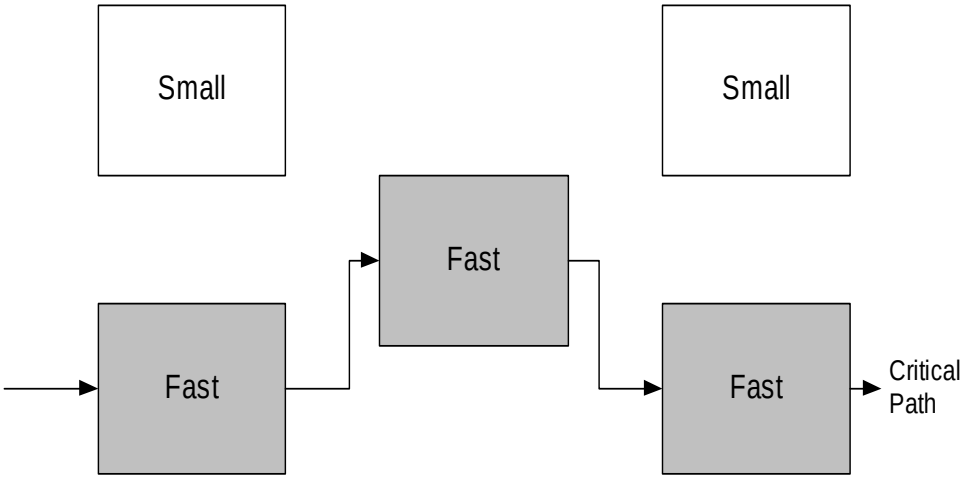
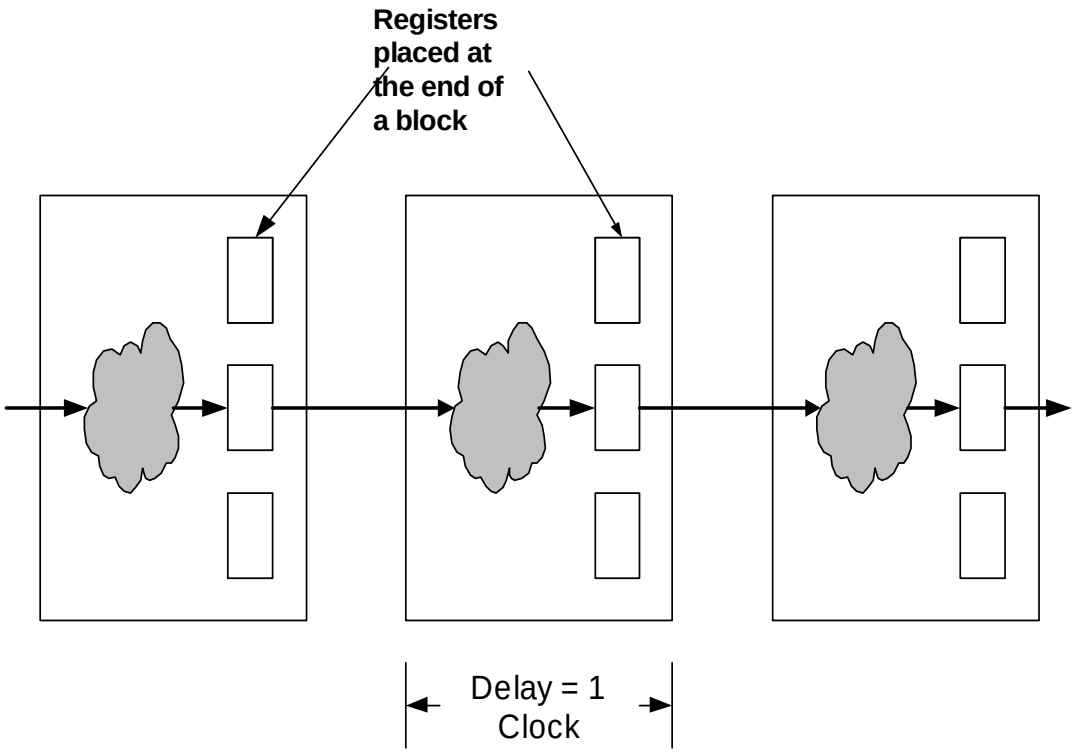


Figure 12-4 Registers Placed at the End of a Block



7. Place registers at the boundaries of hierarchical blocks. There are two barriers that constrain optimization, hierarchical boundaries and registers. Registers are either placed at the front or back of the hierarchical boundaries but not at both front and back! This is recommended when designing hierarchically. Two barriers are combined into a single barrier. This minimizes the impact to overall results when performing bottom-up optimizations. If this design practice is followed then preserving hierarchy in a design has no impact on optimization results and allows for faster CPU run times. Refer to Figure 12-4.

Synthesizing Designs

This section includes:

- State Machine Synthesis
- Setting State Machine Encoding
- Reading Designs

State Machine Synthesis

LeonardoSpectrum encodes state machines during the synthesis process. After a design has been `encoded` during synthesis, the design cannot be re-encoded later in optimization. A well-defined VHDL or Verilog coding style must be followed to allow LeonardoSpectrum to identify the state machine.

Recommended: State machines are isolated into separate hierarchical blocks. This speeds optimization performance and allows for easy modifications to state machine encoding.

Supported State Machine Styles

Binary - Generates state machines with the fewest possible flip-flops. Binary state machines are useful for area critical designs when timing is not an issue.

Gray - Generates state machines where only one flip-flop changes during each transition. Gray encoded state machines are usually without glitches.

Random - Generates state machines using random state encoding. Random state machine encoding should only be used when all other implementations are not achieving the desired results. Random state encoding is not recommended.

OneHot - Generates state machines containing one flip-flop for each state. One hot state machines provide the best performance and shortest clock to out delays. One-hot implementations are larger than binary.

Twohot - Twohot encoding sets two flip flops high for each state. The twohot encoding requires more flip flops than binary and fewer flip flops than onehot. Twohot encoding may be beneficial to large FSMs where onehot uses too many flip flops, and binary requires too much decode logic. Refer to the HDL Synthesis guide, Chapter 2, for more encoding information.

Auto - For auto encoding, LeonardoSpectrum varies the encoding based on bit width. More specifically, enumerated types with fewer elements than global integer `lower_enum_break` are encoded as binary; larger enumerated types are encoded as onehot. Values larger than global integer `upper_enum_break` are encoded as binary. Auto encoding allows LeonardoSpectrum to assign encoding on a case-by-case basis.

Setting State Machine Encoding

There are two ways to instruct LeonardoSpectrum to perform a particular state machine encoding:

- VHDL Attributes or Verilog Pragmas
- Using the LeonardoSpectrum command `set encoding`

Setting VHDL Attributes

To set the encoding for a particular state machine, insert the following statements into your code.

- Declare the `type_encoding_style` attribute. Type `encoding_style` is: (BINARY, ONEHOT, TWOHOT, GRAY, RANDOM, AUTO); attribute TYPE_ENCODING_STYLE: ONEHOT;
- Declare your state machine enumeration type. Type `my_state_type` is: (s0,s1,s2,s3,s4);
- Set the `type_encoding_style` of the state. Type attribute: TYPE_ENCODING_STYLE of my_state_type is ONEHOT;

Setting Verilog Pragmas

To set the encoding for a state machine in Verilog insert the following comment text into your Verilog Model above the state machine model

```
parameter [3:0] // pragma enum state_parameters onehot

idle = 4'b0001,
halt = 4'b0010,
run = 4'b0100,
stop = 4'b1000;

reg[3:0] /*pragma enum state_parameters */state;
```

Note: In the first line of the above code example, the state machine encoding specified is `onehot`. This is an optional specification that could also be set to `binary`, `gray`, and `random`. If the `enum` pragma is specified and not set to a partition, indicate FSM encoding. The encoding default is `onehot` and can be changed with the `set encoding` command.

Setting State Machine Encoding using the `Encoding` *Variable*

Alternatively, the `encoding` variable is used to set state machine encoding. Once this variable is set, all state machines employ the specified encoding until another `set encoding` command issued. Set this variable prior to reading in VHDL or Verilog code

Note: VHDL Attributes and Verilog pragmas override the encoding variable

VHDL Example

```
set encoding onehot

read uart_control_sm.vhdl

set encoding binary

read interface_control_sm.vhdl
```

Verilog Example

```
set encoding binary

read -format verilog control.v
```

Table 12-1. Arguments to the Encoding variable

Arguments	Description
binary	Sets state machine encoding to binary
onehot	Sets state machine encoding to onehot
twohot	Sets state machine encoding to twohot
gray	Sets state machine encoding to grey
random	Sets state machine encoding to random
auto	Sets state machine encoding based on bit width.

Reading Designs

LeonardoSpectrum provides two methods for reading in designs.

`Read` *Command*

The read command analyzes and elaborates the design in one step. Read can be used to input structural or RTL designs in VHDL, Verilog, EDIF, and XNF. Read supports both single file and multi-file designs. Read cannot be used for RTL or for VHDL designs that contain your defined packages. Read cannot be used if you wish to re-define generics during synthesis.

Table 12-2. Arguments to the Read Command

Arguments	Description
-format	vhdl | verilog | edif | sdf | xnf
-dont_elaborate	Only analyze, don't elaborate
-design	Specify the top-level design name to be read
-work	Specify library where read design is to be stored

The analyze and elaborate commands must be used when reading in VHDL design with your defined packages. Verilog synthesis does not require analyze or elaborate. These commands must also be used when re-defining VHDL generics during synthesis.

Table 12-3. Arguments to the Analyze Command

Arguments	Description
-format	vhdl | verilog | edif
-work	Specify library where read design is to be stored

Table 12-4. Arguments to the Elaborate Command

Arguments	Description
-architecture	Root or architecture name
-single_level	Only the top panel of the design
-generics	Redefine specific generic
-parameters	Redefine root level generics
-work	Specify library where read design is to be stored

VHDL Synthesis

Use these steps for VHDL synthesis:

1. Files must be read in bottom-up order, i.e., lower level blocks must be read before the top level blocks.
2. If the design does not contain generics that are passed through hierarchy then use the `read` command exclusively.

```
read bottom.vhd middle.vhdl top.vhdl
```

3. If the design contains generics that need to be re-defined, then use the `elaborate` command. Analyze the packages first, then the VHDL code in a bottom-up order.

```
analyze my_package.vhd

analyze bottom.vhd

analyze middle.vhd

analyze top.vhd

elaborate top -generic data_width = 16
```

Verilog Synthesis

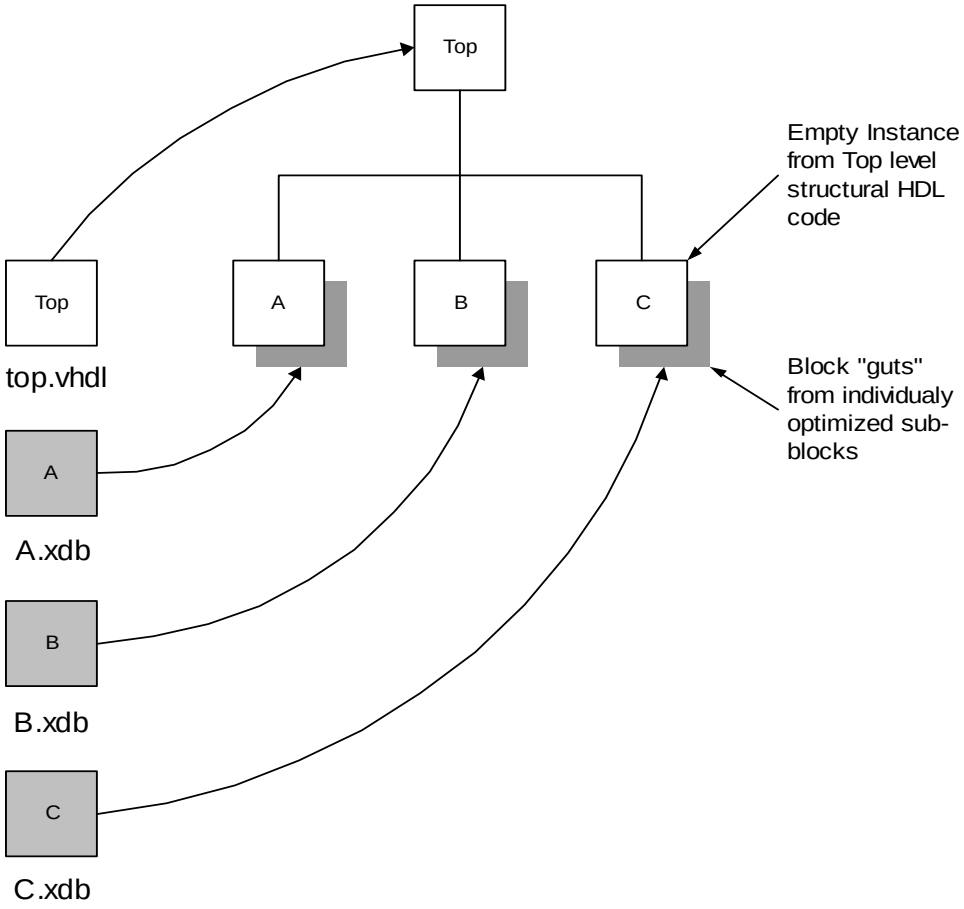
Verilog designs can be read into LeonardoSpectrum in any order. LeonardoSpectrum supports auto-top detection which automatically locates the top-level module so no particular file order is required as in VHDL.

```
read -format bottom.v top.v middle.v
```

Incorporating Structural Blocks

LeonardoSpectrum has the ability to automatically connect sub-blocks with top-level structural netlists - provided all instance names, port names and view names match. The LeonardoSpectrum design browser can be a useful tool when working through design stitching issues. Make sure the target technology is loaded before reading in a structural netlist. Refer to Figure 12-5.

Figure 12-5 Structural Blocks



Designs are stitched bottom-up. This means that all lower level blocks, that have completed optimization, are read into LeonardoSpectrum first. The next step is to synthesize the top-level structural VHDL or Verilog file which connects the sub-blocks together. This requires that all instance names and port names match except for case.

```
read -format xdb {A.xdb B.xdb C.xdb}
```

```
read -format {vhdl top.vhdl}
```

Note: View names between the sub-blocks and the instances contained within the top-level structural code must match exactly for bottom-up design stitching to be successful.

Adding Cells and Modeling Memories

This section includes:

- Inserting RAMs Through Synthesis
- Using Xlib Creator to Add RAM Cells
- Optimizing with Auxiliary RAM Cells

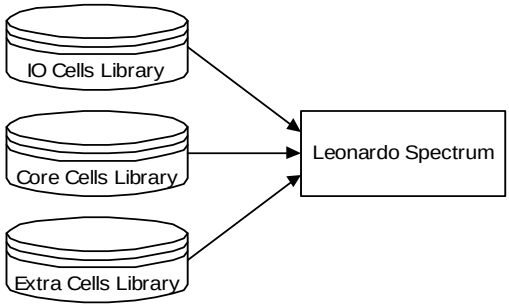
Inserting RAMs Through Synthesis

LeonardoSpectrum has the ability to infer RAMs from RTL code for programmable logic devices. For an ASIC design, you must instantiate RAM cells directly into the netlist.

Using XlibCreator to add RAM cells

When designing an ASIC with RAMs, an auxiliary cell is created that contains a timing model of the RAM cell. This second library can be loaded into LeonardoSpectrum along with the core library. Refer to Figure 12-6.

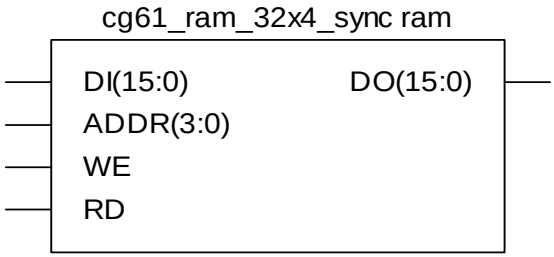
Figure 12-6 Auxiliary cells



Modeling RAMs in Libgen format

Modeling RAMs in Lgen format requires the definition of the following fields:

Figure 12-7 Modeling RAMs



Library

Set to the same value as the core library technology. For example, the following syntax is used for the Fujitsu cg61 library.

LIBRARY cg61

Gate

The next step is to define the RAM cell and pin list. For example, assume that the RAM is called cg61_ram_32x4_sync. Define this RAM as follows:

```
GATE cg61_ram_32x4_sync

Inputs DI3, DI2, DI1, DI0;

Inputs ADDR3, ADDR2, ADDR1, ADDR0;

Inputs WE, RD;

Outputs DO3, DO2, DO1, DO0;
```

Special Properties

There are four special properties that must be added to the RAM model in the following order:

- ***nomap*** - The “SAME_TECH_NOOPT” variable is set to allow component instantiation of the cell.
- ***function ()*** **–** The function parameter allows you to define a particular function for a gate. This is not necessary for RAM models which are treated as black-boxes.
- ***area*** **–** Specify the RAM cell area in gates. LeonardoSpectrum uses this information during area reporting.
- ***SET SAME_TECH_NOOPT*** - The `SAME_TECH_NOOPT` variable must be set to allow component instantiation of the RAM cell.

```
nomap;
function ();
area = 5000;
SET SAME_TECH_NOOPT;
```

Electrical Information

Electrical information refers to the definition of input capacitance loading and output drive characteristics. These fields must be defined correctly to allow LeonardoSpectrum to adjust buffering around the RAM cell. Define electrical information on a pin-by-pin basis as follows:

Input DI1 (cap_load = 0.024862);

Output DO1 (max_fanout_load = 0.273485);

cap_load **–** Specified on input pins of a gate to indicate the input capacitance of this pin. Load is expressed in terms of pf.

Default_cap_load **–** Default value for cap_load. This value is assigned to all input pins that do not have a cap_load value

Max_cap_load **–** Specified on output pins of a gate to indicate the maximum capacitance that can be driven by this pin

Default_max_cap_load **–** Default value for max_cap_load. This value is assigned to all output pins that do not have a cap_load value.

Fanout_load **–** Gives the fanout load value for an input pin. The sum of all fanout_load values for input pins connected to a driving output pin must not exceed the max_fanout value for that output pin. There are no fixed units for fanout_load; typical units are standard loads or pin count.

Default_fanout_load – Default value for fanout_load for all input pins in a design which do not have a fanout_load specified

Max_fanout_load – Defines the maximum fanout load that an output pin can drive. LeonardoSpectrum performs buffering and logic replication to correct any fanout_load violations

Default_max_fanout_load – Default value for max_fanout load for all output pins in a design which do not have a max_fanout_load specified.

Max_transition – The max_transition value defines a design-rule constraint for the maximum acceptable transition time of an input or output pin. If max_transition is used with an output pin, then that pin can be used only to drive a net. The cell can then provide a transition time at least as fast as the defined limit. A max_transition value used with an input pin indicates that the pin cannot be connected to a net that has a transition time greater than the defined limit.

Default_max_transition – Sets the default value for max_transition for all output pins in the library that do not have a max_transition values.

Delay Information

This is the most complex element part of the RAM cell definition. Each timing arch must be defined. Any address or data input can effect the delay to any output data pin while the device is in read mode. In the example, there are 32 timing arcs that must be defined. Since these timing arcs are all identical, cut and paste can reduce the workload dramatically. Delays can be modeled as integer values, linear equations or lookup tables. In this example, the method is to use the ***prop*** (propagation) property to define an integer value for the input to output delay.

Note: The ***prop*** field requires two values: rise and fall time. More complex modeling methods exist including piecewise linear and lookup table definitions. Setup times are defined for the `we` and `rd` pins as follows:

```
Input we (
Delay clk (
Setup = (1.5, 1.7);)
);

Input addr0 (
Delay do0 (
Prop = (3.1, 3.5);)
);
```

Rules

- Input to output delay arcs are specified on input pins only when using the ***delay*** keyword. No delay specification is required on output pins.
- Input pins that require a setup must reference the clock through the ***delay*** keyword
- The ***setup*** and ***prop*** properties require two fields: rise time and fall time.

Note: Any path can be disabled by inserting the line `Path_type = DISCONNECT` within the delay specification.

Example Lgen RAM model

```
LIBRARY cg61 (
   GATE ram_a1 (
Inputs DI1, DI0, ADDR1, ADDR0, WE, RD;
outputs DO1, DO0;

nomap;              # add for this cell is non-mappable
function ();
area = 5000;
SET SAME_TECH_NOOPT;    # add for specific instantiation

input DI1 ( cap_load = 0.02; );
input DI0 ( cap_load = 0.02; );
input ADDR1 ( cap_load = 0.02; );
input ADDR0 ( cap_load = 0.02; );
input WE ( cap_load = 0.02; );
input RD ( cap_load = 0.02; );
output DO1 ( max_fanout_load = .3; );
output DO0 ( max_fanout_load = .3; );
input DI1 ( delay DO1 ( prop = (3.1, 3.5); ));
input DI1 ( delay DO0 ( prop = (3.1, 3.5); ));
input DI0 ( delay DO1 ( prop = (3.1, 3.5); ));
input DI0 ( delay DO0 ( prop = (3.1, 3.5); ));
input ADDR1 ( delay DO1 ( prop = (3.1, 3.5); ));
input ADDR1 ( delay DO0 ( prop = (3.1, 3.5); ));
input ADDR0 ( delay DO1 ( prop = (3.1, 3.5); ));
input ADDR0 ( delay DO0 ( prop = (3.1, 3.5); ));
input WE (delay DI1 (setup = (0.5, 0.7); ));
input WE (delay DI0 (setup = (0.5, 0.7); ));
   );
)
```

Using Lgen

Lgen is a licensed library builder tool. Lgen can be used without a license to create non-mappable libraries, which is exactly what a RAM or IP model is. From either a DOS or UNIX shell type the following command:

```
> lgen <ram_lgen_sourcename.lgn> <technology_name_ram.syn>
```

```
> lgen cg61_ram32x4.lgn cg61_ram.syn
```

Loading Auxiliary Libraries into LeonardoSpectrum

LeonardoSpectrum supports loading multiple libraries. Load the auxiliary library along with the core library prior to optimization. Before loading the library, place the library in the $EXEMPLAR/lib directory along with the core library in any order.

```
>load_library cx2001_rams
```

```
>load_library cx20001
```

Optimizing with Auxiliary RAM cells

You should apply a `dont_touch/noopt` attribute to all instantiated RAMs and auxiliary cells. This ensures that optimization does not remove the RAM cells

```
> NOOPT lib_name.top_level_cellname.instance_name.ramcell_instname
```

```
> NOOPT work.cx2001_rams.ram_dual_1024_4
```

Setting Constraints

Constraints in LeonardoSpectrum can be either as simple as specifying the target design frequency or as powerful as indicating multi-cycle paths between flops. Timing constraints indicate desired target arrival and required times used for setup and hold analysis. Constraints are applied after the design is read into LeonardoSpectrum and before optimization. LeonardoSpectrum assumes intuitive defaults. At a minimum, you must define the clock, input port arrival times, and output port required times.

Note: LeonardoSpectrum does not support timing constraints relative to a particular clock. Everything is referenced to time zero. This includes input arrival times and output setup times without any reference to a particular clock.

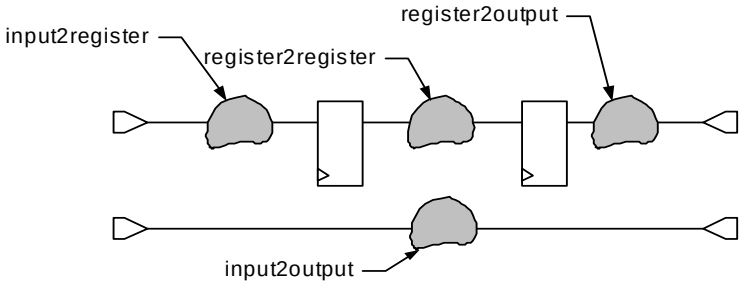
This section includes:

- Global Timing Constraints
- Clock Constraint
- Clock Skew
- Clock Uncertainty
- Input Arrival Time
- Output Required Times
- Multicycle Path Constraints
- False Path Constraints
- Constraining Purely Combinatorial Designs
- Constraining Mixed Synchronous and Asynchronous Designs

Global Timing Constraints

Global timing variables, similar to other global variables, apply to the present design in memory. Explicitly defined timing constraints override global constraints. Setting these variables saves considerable time and effort when performing bottom-up optimizations. Refer to Figure 12-8.

Figure 12-8 Global Timing Constraints



Hint: Set the `input2register` and `register2output` variables to one-half of the clock period. This ensures that the boundary logic of subblocks meets timing when combined into the top-level design.

```
set register2register 20
set input2register 10
set register2output 10
set input2output 10
```

Resets

Sequential element set and reset pins are automatically blocked during timing analysis. No special settings or constraints are required to block reset timing paths.

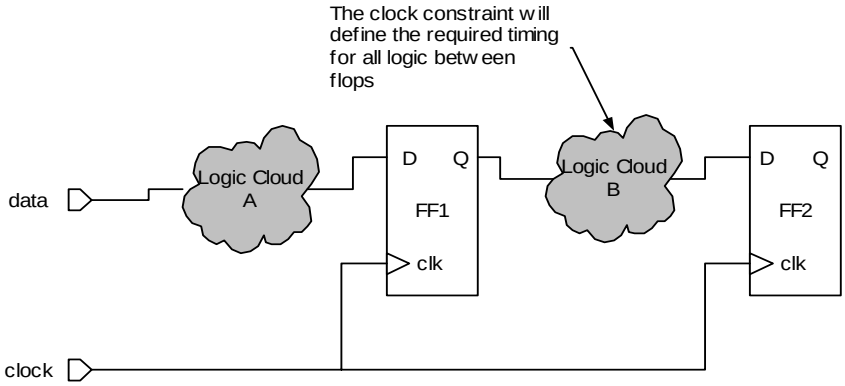
Clocks

LeonardoSpectrum supports one or more synchronous clocks. However, multiple asynchronous clocks are not supported.

Clock Constraints

Clocks define timing to and from registers. Without clocks defined, all registers are assumed unconstrained. Therefore all combinational logic between registers is ignored during timing optimization. When you define a clock, you have effectively constrained the combinational logic between all registers to one clock period. Refer to Figure 12-9.

Figure 12-9 Clock Constraints



The logic between FF1 and FF2 is constrained to one clock period. If clock period is 50ns, then Logic Cloud B has approximately 50ns – setup of FF2 to meet timing. LeonardoSpectrum describes clocks by using three basic commands:

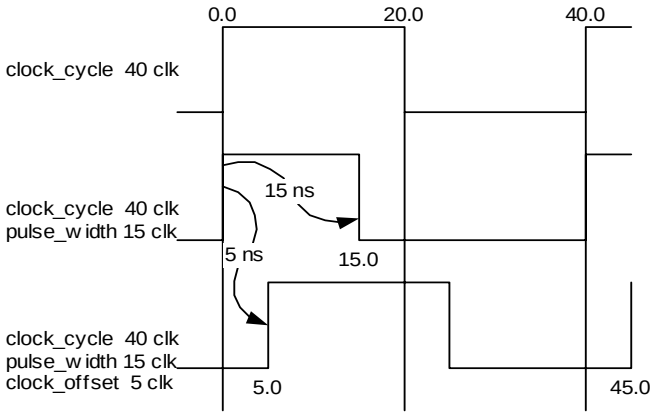
```
clock_cycle <clock_period> <primary_input_port>
pulse_width <clock_pulse_width> <primary_input_port>
clock_offset <clock_offset> <primary_input_port>
```

By default, the clock network is assumed to be ideal - with no clock delay. The clock arrives at the same time between all flops. To change clock network to propagated delay, set `propagate_clock_delay` variable to true.

Example of clock constraints:

In the first example in Figure 12-10, clock period is defined as 40 ns and attached to clock port “clk”. The default duty cycle is 50%, or a clock pulse width of 20ns. The second example shows how to change the pulse width to 15ns. The third example demonstrates how one can offset the clock. This is useful for specifying a clock skew relative to zero.

Figure 12-10 Clock Network



Clock Skew

When constraining a design you may want to accommodate clock skew. Clock skew is often the result of a clock delay incurred by the input clock driver. This may effect the offsetting of the clock by the skew value. LeonardoSpectrum does not provide a direct method for the input of clock skew, however the clock may be offset by the skew value to create the same effect for timing analysis and optimization.

Figure 12-11 Clock Skew

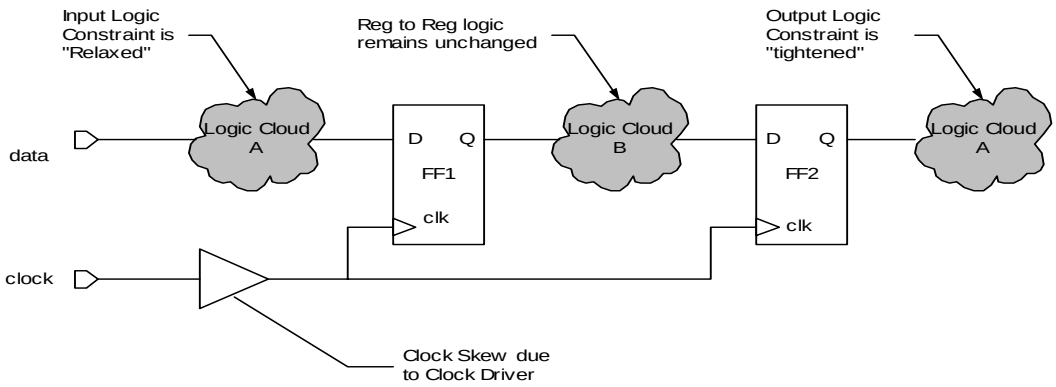
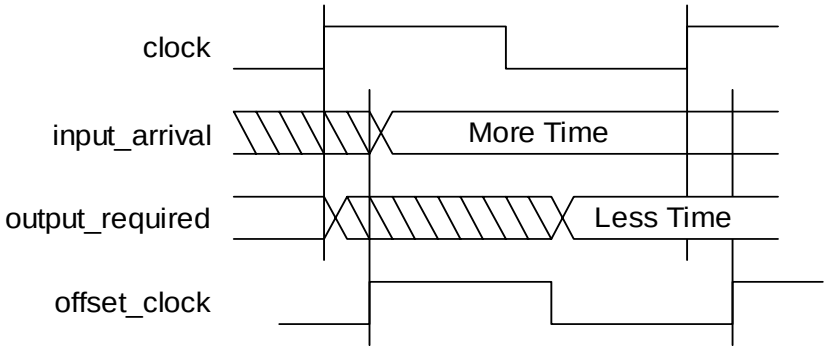


Figure 12-12 Clock Skew Timing

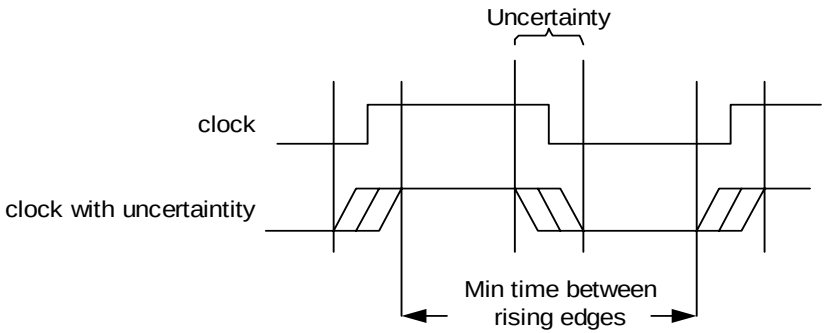


Setting a clock skew reduces the input arrival time by the skew value. This provides a *relaxed* constraint for optimization of that logic. Clock skew does not effect register-to-register logic since all flops are subject to the same skew. Clock skew reduces the output-required time which *tightens* the timing constraint on the output logic by the skew value.

Clock Uncertainty

ASIC Vendors often specify clock uncertainty values for their devices. This has the effect of over constraining a design to account for clock signal variations. Refer to Figure 12-13.

Figure 12-13 Clock Uncertainty



Use these steps to set a constraint in Leonardo Spectrum that accounts for clock uncertainty:

1. Subtract 2*(Clock Uncertainty) from the clock period. For example if the clock period was 10 ns and the uncertainty was 1 ns than the worst case condition would occur if the clock was 1 ns late on a rising edge immediately followed by a rising edge that was 1 ns early. This creates a worst case condition of 8 ns.
2. Perform Timing optimization and timing analysis with the report_delay_analysis_mode variable set to minimum (default).
3. Add 2*(Clock Uncertainty) to the clock period. Using our previous example the clock period would now become 12.
4. Set the report_delay_analysis_mode variable to maximum. This will allow you to perform a worst-case hold time analysis. Re-run timing analysis and fix any hold time errors.

Multiple Synchronous Clocks per Block

The timing analyzer for LeonardoSpectrum supports only 1 clock per block for exhaustive timing analysis. Designs with multiple synchronous or asynchronous clocks can be analyzed using a technique involving the clock offsets. Refer to Figure 12-14, where Block A and Block B are each driven by different, synchronous clocks.

Figure 12-14 Multiple Synchronous Clocks

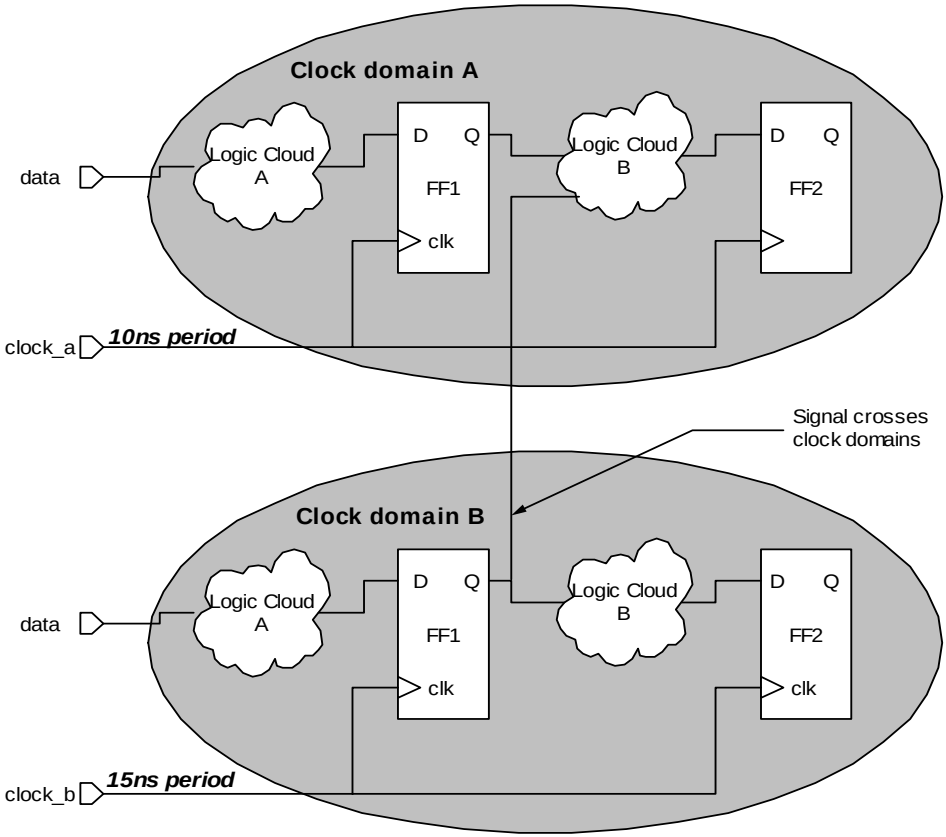
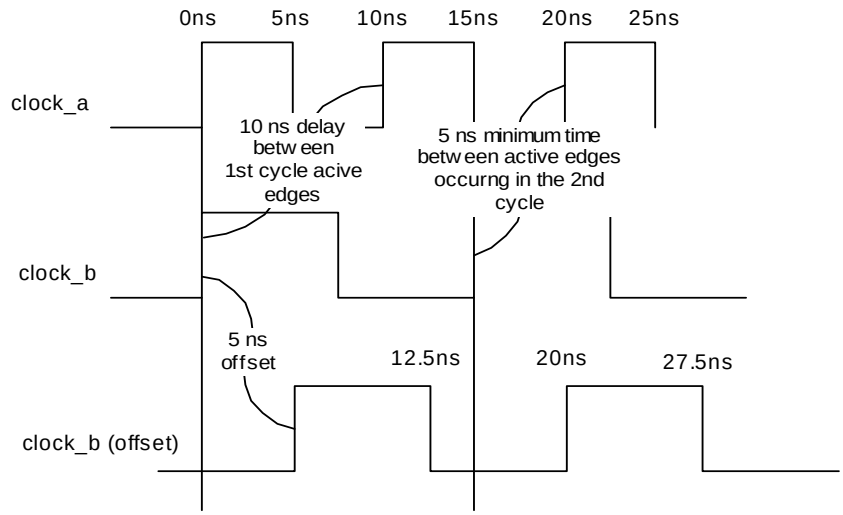


Figure 12-15 Synchronous Clocks



Procedure for setting multiple, synchronous clock constraints

1. Draw the clock waveforms starting from time zero and complete several clock cycles. Manually determine the minimum time between active edges - this may not occur in the first clock cycle - this depends on how the active edges meet. In Figure 12-15, the minimum time is 5 ns.

2. Determine what the time between active edges is during the first clock cycle. The delta is 10 ns in Figure 12-15.

3. Subtract the minimum active edge time from the first cycle active edge time. This number becomes the clock offset for the clock of signal origin. Set the appropriate clock offset:

```
> clock_offset 5 clock_b
```

4. **CAUTION**: Setting the clock offset alters the input arrival timing. If you set an input arrival time of 6 for example, then the clock offset has essentially added that number to the offset (6 + offset). You now have to adjust the input arrival time to correct for the offset by adding the offset to the input arrival.

Multiple Asynchronous Clocks

LeonardoSpectrum does not analyze timing for signals that cross between two or more asynchronous clock domains. This is because the clocks do not have a defined relationship. The best way to handle this is to ignore all timing between signals that cross between asynchronous clock boundaries. This can be accomplished by assigning a clock offset to the clock of signal origin that is equal to or greater than two clock periods. To disable timing between clock boundaries in Figure 12-16 issue the following command:

```
> clock_offset 30 clock_b
```

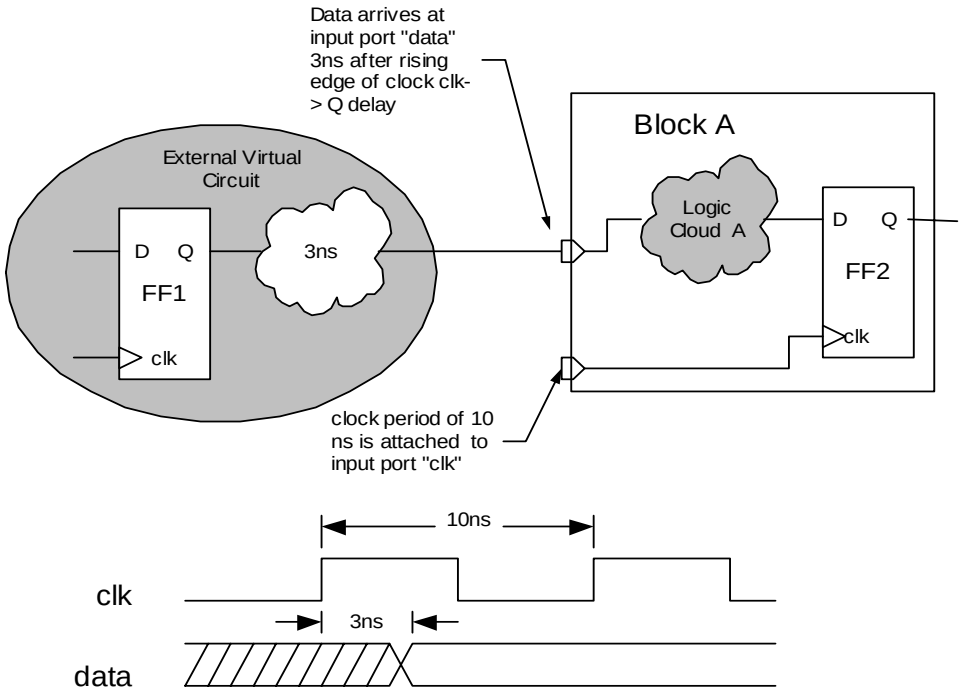
Input arrival times, if any, need to be increased by the amount of the clock offset.

Input Arrival Time

The input arrival time specifies the maximum delay to the input port through external logic to the synthesized design.

```
arrival_time <delay_value> <input_port_list>
```

Figure 12-16 Input Arrival Time



In Figure 12-16, data arrives approximately 3 ns after the rising edge of clock. Therefore, to accurately constrain the input port data, you must apply the following constraint:

```
> arrival_time 3 { data }
```

If the clock period were defined as 10ns, then the setup of FF2 must be added to the combinatorial delay of logic cloud A, and needs to be 7ns to meet timing.

Note: All input arrival times start at time zero and cannot be specified relative to a particular clock edge. To adjust for a particular clock edge, you must add the clock offset to the arrival time.

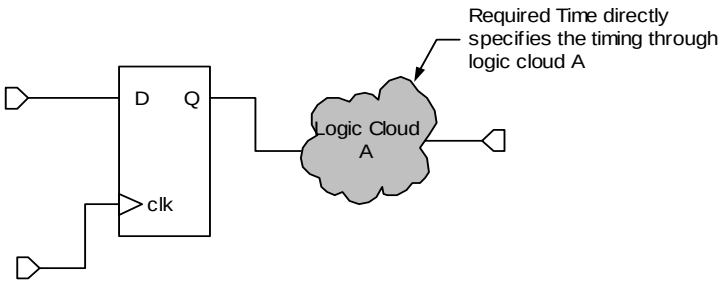
Output Required Times:

The output required time specifies the data required time on output ports. Time is always with respect to time zero. In other words, output required time cannot be specified relative to a particular clock edge

```
> required_time <required_value> <output_port_list>
```

Refer to Figure 12-17.

Figure 12-17 Output Time



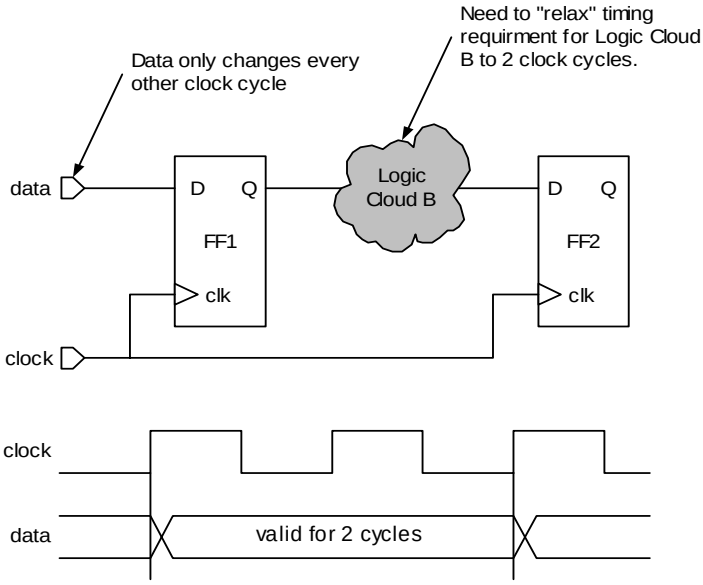
When specifying required times all constraints are assumed to begin at time zero. This eliminates the need to specify a constraint relative to a particular clock edge. The specified required time becomes the time constraint on the output logic cloud shown as logic cloud A in Figure 12-17.

```
> required_time 7 { d1 }
```

Multicycle Path Constraints:

Leonardo, version 4.2, and LeonardoSpectrum offer the ability to constrain individual paths to more than one cycle. Refer to Figure 12-18.

Figure 12-18 Constrain Individual Paths



To appropriately constrain the design in Figure 12-18, apply the following constraints:

```
> set_multicycle_path -from {FF1} -to {FF2} -value 2
```

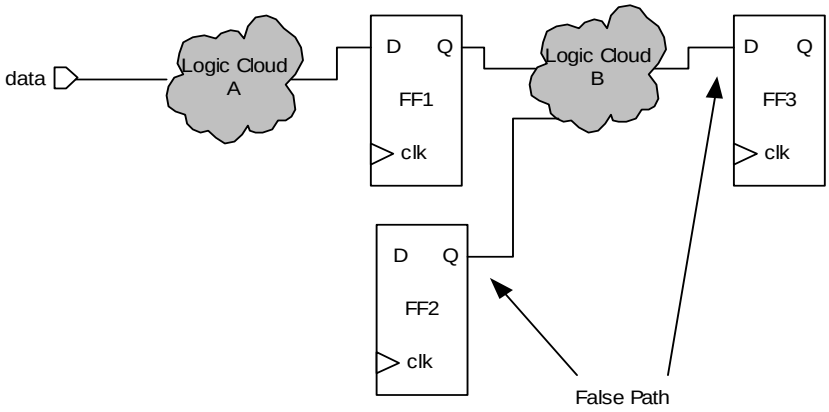
This constraint has the effect of setting logic cloud B to 2 clock periods minus the setup of FF2.

Caution: Use caution when using multi-cycle constraints since timing analysis is slowed. A few multi-cycle constraints may have little effect; however, many multi-cycle constraints may slow timing optimization considerably.

False Path Constraints:

False paths are design paths that you want LeonardoSpectrum to ignore for timing optimization. Refer to Figure 12-19:

Figure 12-19 False Path Constraints



By taking advantage of the multi-cycle command, you can specify the path from FF2 to FF3 as false. Refer again to Figure 12-19:

```
> Set_multicycle_path -value 1000 -from {FF2} -to {FF3}
```

Essentially, the path from FF2 to FF3 has been constrained to 1000 clock cycles. Since logic cloud B probably would not ever take more than 1000 cycles, this path has effectively been eliminated from timing optimization and timing analysis.

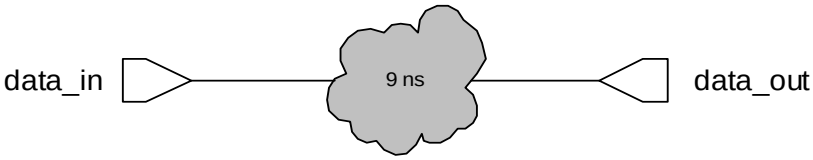
Caution: Apply the same caution to false paths as applied to multi-cycle constraints. A few false paths may have little effect; however many false paths may increase timing analysis run times.

Constraining Purely Combinatorial Designs

A purely combinatorial design contains no clocks. You can constrain these blocks by specifying the global variable `input2output`. This constrains any purely combinatorial paths through a circuit. For example:

```
set input2output 9
```

Figure 12-20 Combinatorial Design



Constraining Mixed Synchronous and Asynchronous Designs

Some blocks have both synchronous and purely combinatorial paths through the circuit. A mealy state machine is a good example of this. To constrain these designs, you apply synchronous constraints to the ports of the synchronous paths and asynchronous constraints to the ports of the asynchronous paths.

Figure 12-21 Constraining Designs with Mixed Signals (1 of 3)

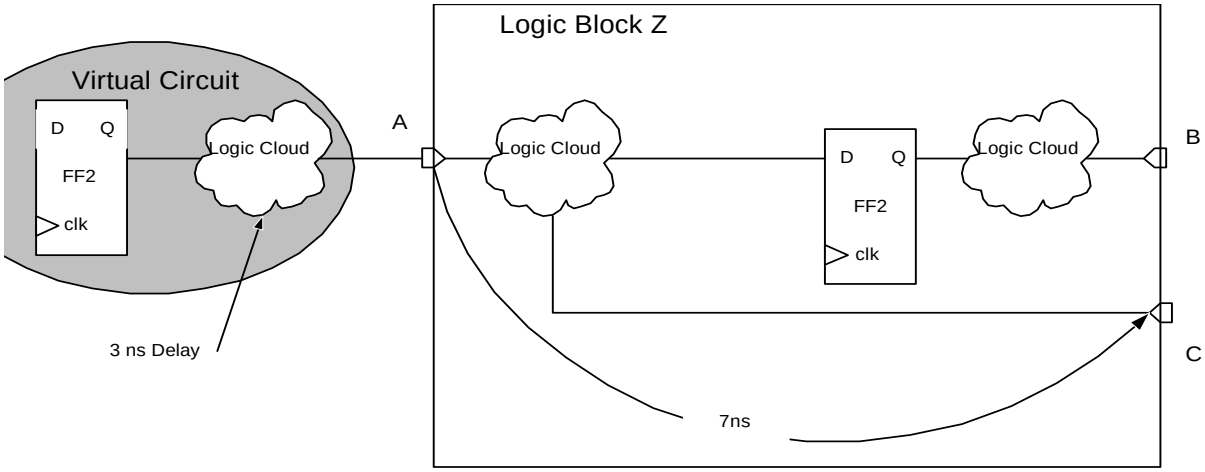


Figure 12-22 Constraining Designs with Mixed Signals (2 of 3)

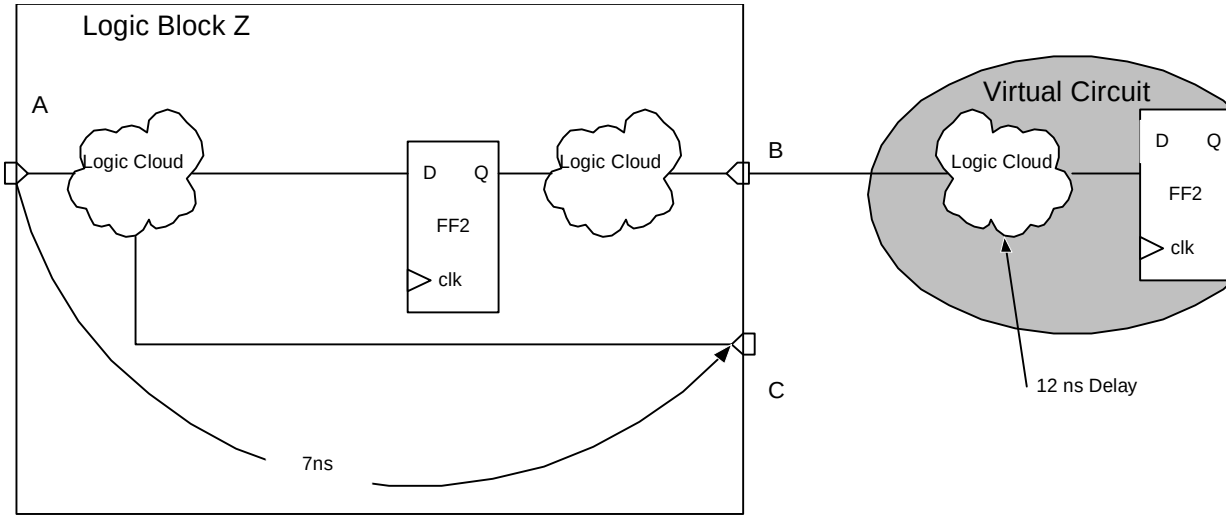
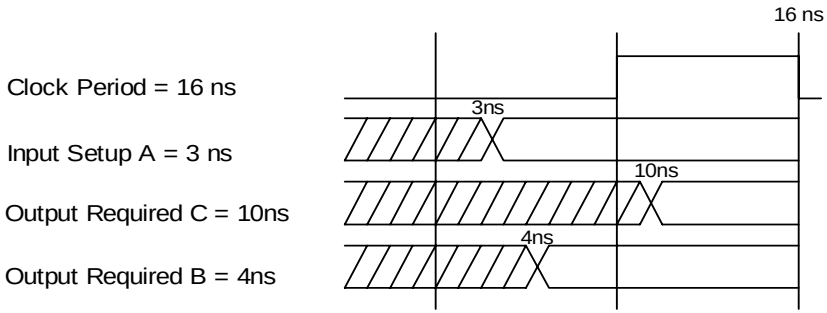


Figure 12-23 Constraining Designs with Mixed Signals (3 of 3)



Procedure for Setting Constraints on Mixed Designs

1. Define the clock constraints. Refer to Figures 12-21, 12-22, and 12-23.

```
> clock_cycle 16 clk
```

2. Apply an input arrival constraint assuming the design is entirely sequential.

```
> arrival_time 3 A
```

3. Apply an output-required time to the sequential output ports only. Set the constraints for a sequential circuit ignoring the combinatorial paths for now.

```
> required_time 4 B
```

4. Apply an output arrival time to the combinatorial output paths. The maximum delay constraint applied to these paths is the window created by the difference between the input arrival time and the output-required time. In Figure 12-23, an input arrival of 3 is set and a maximum delay through the combinatorial path of 7ns is desired. The output required time must be 10ns (10ns - 3ns = 7ns).

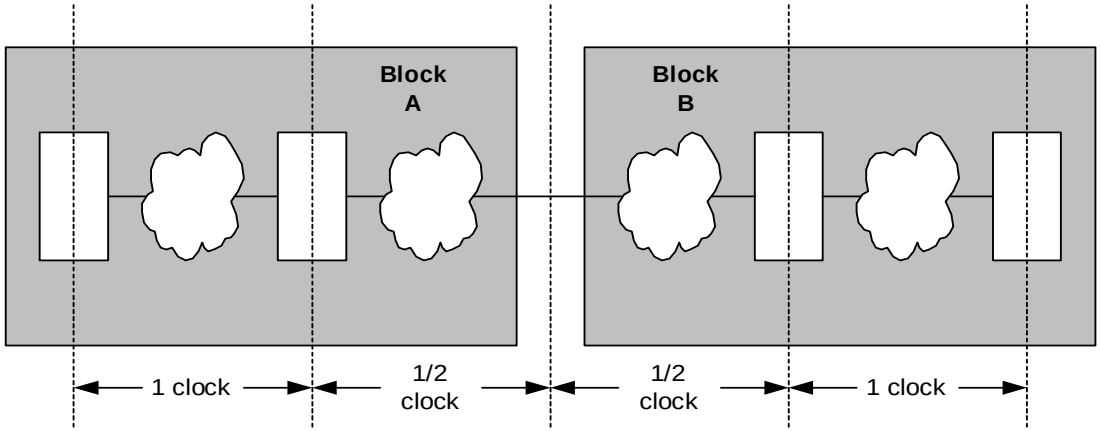
```
required_time 10 C
```

Constraining Sub-blocks for Timing

Ideally, registers are placed at hierarchical boundaries. However, random logic can be placed at the hierarchical boundaries, which forces you to constrain the logic appropriately. Unless more detailed information about the sub-block timing is known, use constraints that equal to one-half of the clock period for application to the boundary.

If both sides meet timing, then when the blocks are combined, timing is met. Define the global register2output and input2register variables to equal one-half of the clock period. Refer to Figure 12-24.

Figure 12-24 Constraining Sub-Blocks

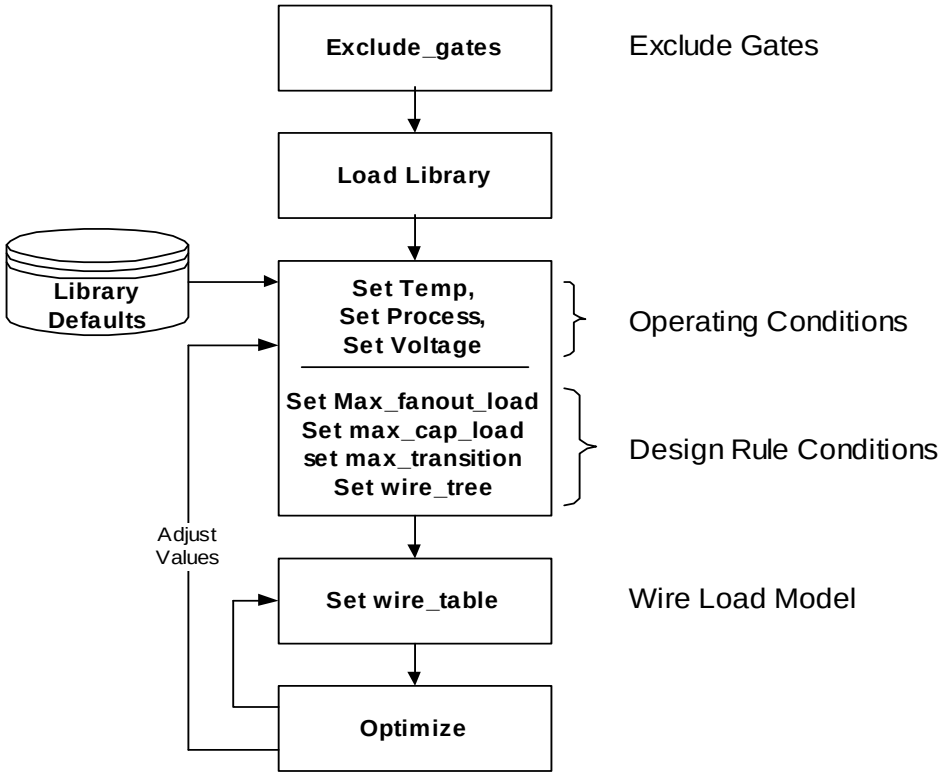



```
clock_cycle 20 clk
set input2register 10
set register2output 10
```

Global Optimization Variables

As shown in Figure 12-25, global variables apply to the entire design in memory which is referred to as the present design. To ensure accurate timing analysis, you must set the variables temperature, voltage, and process before loading the library. These variables are used to calculate a scaling factor that slightly adjusts the timing values for library cells. This scaling calculation takes place during library load and cannot be modified. These values can be dynamically modified at any time.

Figure 12-25 Global Variables



Operating Conditions

Operating conditions specifies the environment at which the ASIC is operating. This includes the external environment conditions of temperature, process, and voltage. These variables have the effect of scaling the delay calculation values.

Temperature:

Temperature is set to an integer value. Values range from 55 degrees to 95 degrees. This value is typically set to 80 degrees. Use the following command to set temperature.

```
> set temp 80
> show_var_settings
Set temp “70”
```

Voltage:

Sets the operating voltage for the design. Values depend on the voltage level of the silicon: 5 volts or 3.5 volts. Use the following command to set temperature.

```
> set voltage 5
> show_var_settings
Set voltage “5”
```

Process:

The process variable is set to a `min`, `typical`, or `max` process. The library developer defines the actual character string used. Some ASIC vendors create separate “min”, “typ” and “max” libraries. In this example, setting this variable has no effect. In other examples, nine processes are defined that are typically labeled “wccom”, “typcom”, “bccom”, “wcind”, “typind”, “bcind”, “wcmil”, “typmil” and “bcmil”. Refer to the .doc file supplied with the design kit for the exact process syntax. Use the following command to set the process variable.

```
> set process TYPICAL
> show_var_settings
Set process “TYP”
```

Note: The only way to view the default values of `temp`, `process`, and `voltage` is to generate a timing report. The current settings are listed at the top of the report. The default values for the operating condition variables max_fanout, max_cap_load,

max_transition, wire_table and exclude_gate are not available in the LeonardoSpectrum GUI. Please refer to the vendor-supplied documentation for this information.

Design Rule Conditions

The final optimization LeonardoSpectrum performs on a circuit is referred to as `balance_loads`. This algorithm resolves design rule violations by adjusting the circuit through logic replication, buffer insertion and driver sizing. Operating conditions define the fanout, loading and transition thresholds for nets where buffering takes place. The `balance_loads` command is executed automatically from the LeonardoSpectrum primary optimization commands: `optimize` and `optimize_timing`; `balance_loads`.

Fanout:

Globally specifies the maximum fanout on a net

```
> set max_fanout_load 16
```

Capacitance Load:

The maximum allowable capacitance on a net is globally specified. Capacitance on a net is defined as the sum of the route table capacitance plus the accumulated totals of the input pin loading capacitance. Refer to Figure 12-26.

```
> set max_cap_load 2
```

Note: `optimize_drc_resolving`

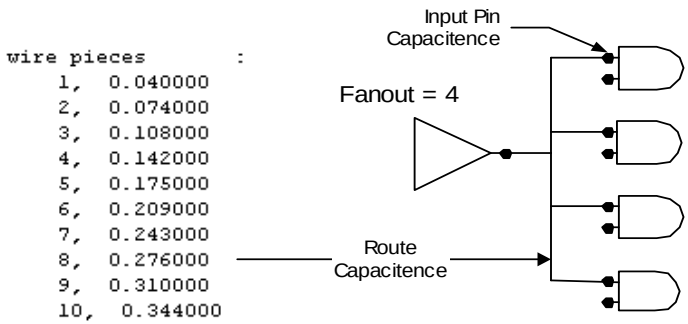
Enables DRC (design rule checking) resolving during optimization by default. Default value: TRUE.

For example, if you are using script for an ASIC design, then you can `set optimize_drc_resolving false` to disable this variable.

Note: You must run the `balance_loads` command at the end of your design run to ensure that the final design meets the design rule checking (DRC).

Note: DRC resolving may require more runtime. However, DRC resolving can improve the initial timing estimation, and can prevent heavily loaded nets.

Figure 12-26 Capacitance Load

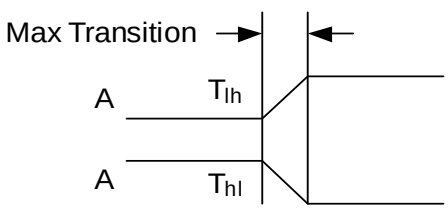


Total Net Capacitence = Route Cap+ Total Input Pin Cap

Transition Time:

Specifies the worst case transition time on a net. Refer to Figure 12-27.

Figure 12-27 Transition Time

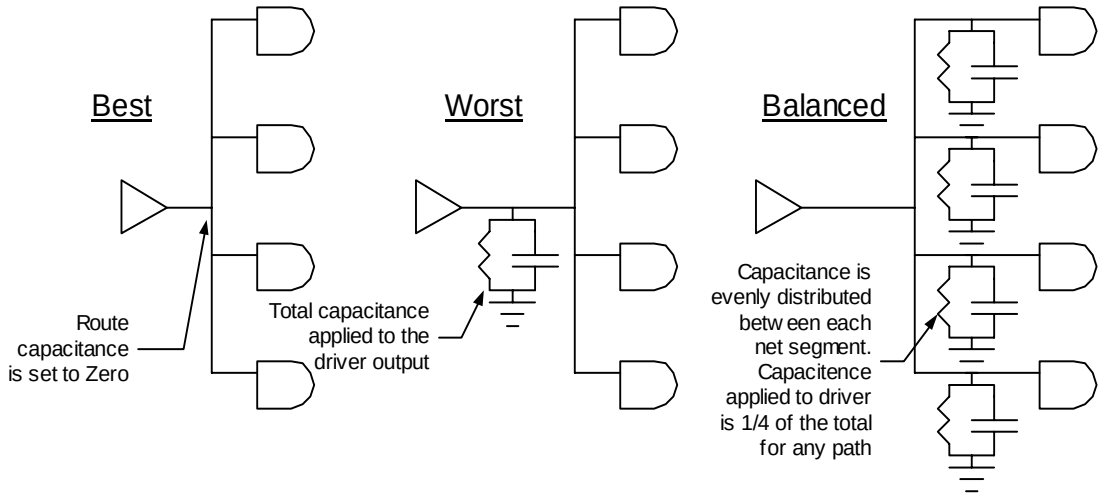



```
> set max_transition .5
```

Wire Tree

Three different models of the capacitance load can be used: best, balanced and worst. In the best case the interconnect delay is 0, balanced case divides the capacitance evenly between the driven loads and the worst case the total net capacitance is lumped into a single value. Refer to Figure 12-28.

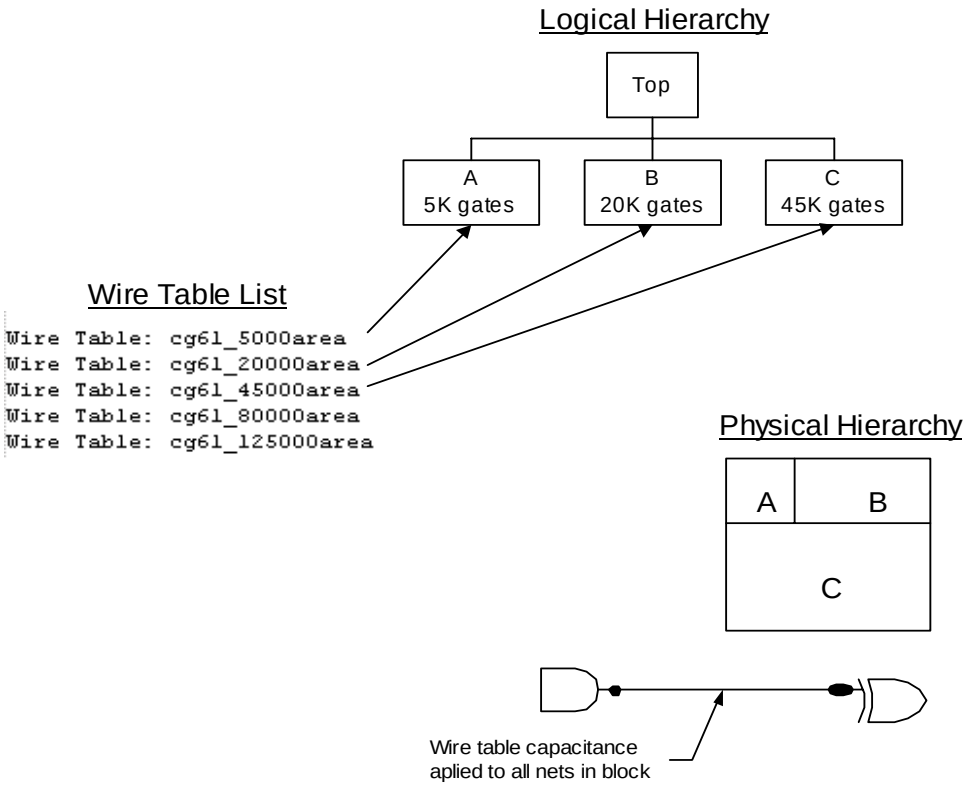
Figure 12-28 Wire Tree


Best
Worst
Balanced
Route capacitance is set to Zero
Total capacitance applied to the driver output
Capacitance is evenly distributed between each net segment. Capacitence applied to driver is 1/4 of the total for any path

Wire Load Model

LeonardoSpectrum supports individual wire load models for individual blocks. Based on estimated routing capacitance, the wire load model defines one component of the total net capacitance. Wire load models are assigned based on the estimated gate area of the physical hierarchy blocks. Refer to Figure 12-29.

Figure 12-29 Wire Load Model

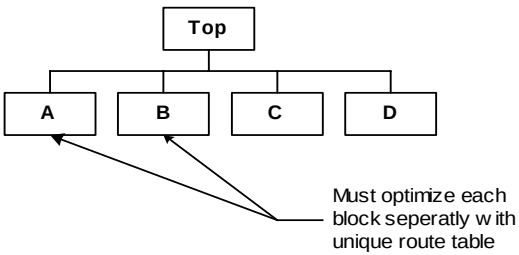


Different wire load models are applied to the individual subblocks of a design that correspond to the physical hierarchy. Low-level hierarchical blocks are often “dissolved” into higher level blocks during floor planning. Use the following command to set the wire table for a block:

```
> report_wire_tables -summary
> set wire_table cg61_5000area
```

Note: Route tables can only be applied to the `present_design` and once applied effect the entire `present_design`. If sub-blocks require different route tables, then these tables must be optimized separately to ensure correct buffering and accurate timing analysis. Use a script similar to the following. Refer to Figure 12-30.

Figure 12-30 Blocks for Optimizing




```
> present_design A
   > Set wire_table table1
   > optimize
   > report_delay
   > present_design B
   > set wire_table table2
   > optimize
   > report_delay
```

Use the `report_wire_tree` command to view the available wire load models as defined in the library

Table 12-5. Arguments to the report_wire_tree command

Arguments	Description
-library	Report wire trees for a specified library
-summary	Report a brief summary of wire tree information
-details	Report wire tree information in complete detail

Exclude Gates:

Any register or boolean gate can be excluded to prevent use of that gate during mapping and optimization. This is set before performing optimization. Exclude gates does not remove a gate, after the gate is inserted in the netlist. To exclude gates use the following command:

```
set exclude_gates {gate1 gate2 gate3}
```

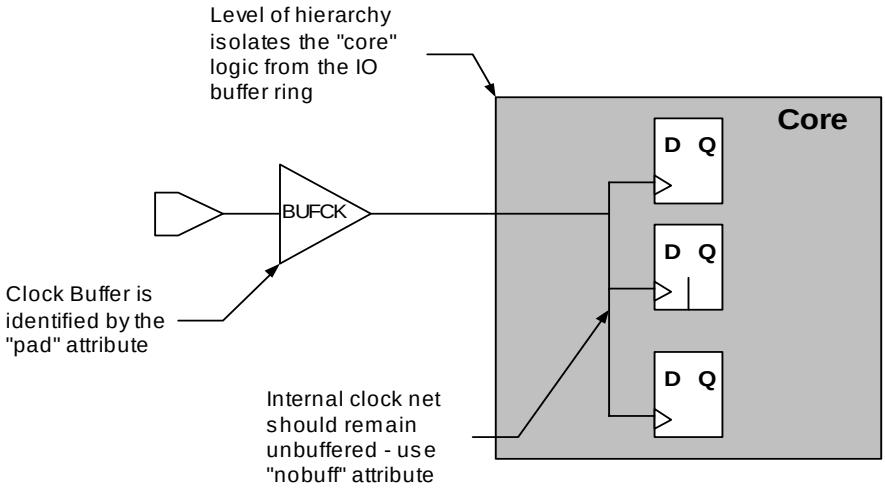
Clock Buffering

ASIC Vendors typically have two requirements for clock lines. Refer to Figure 12-31.

1. No buffer trees are inserted on the internal clock signal.

2. A special IO buffer is used to identify a port as a clock signal.

Figure 12-31 Clock Buffering


Level of hierarchy isolates the "core" logic from the IO buffer ring
Core
D Q
D Q
D Q
BUFCK
Clock Buffer is identified by the "pad" attribute
Internal clock net should remain unbuffered - use "nobuff" attribute

Leaving Clock Lines Unbuffered

Use the `nobuff` command in LeonardoSpectrum to leave the clock net unbuffered during optimization. The `nobuff` command sets an attribute on the clock net that prevents buffering. Buffers cannot be removed after insertion. For this reason `nobuff` is executed before performing any optimization commands.

```
nobuff port clock
optimize –ta cg61 –macro -area
```

Inserting the Clock IO buffer

Clock IO buffers are inserted together with all other chip IO. Currently, LeonardoSpectrum does not allow automatic IO insertion which leaves the clock signal without a buffer. In this situation, component instantiation is used. When a specific buffer is required, use the `pad` command to identify the specific buffer.

```
pad clock BUFCK
```

The `optimize` command must be used to insert buffers. The `optimize` command is also used to perform internal load balancing. Once a buffer is inserted the internal net, the `nobuff` attribute is lost and the buffer is then subject to load balancing. To prevent this, group the core logic into a single block of hierarchy, insert the buffers using optimize, then ungroup the hierarchy.

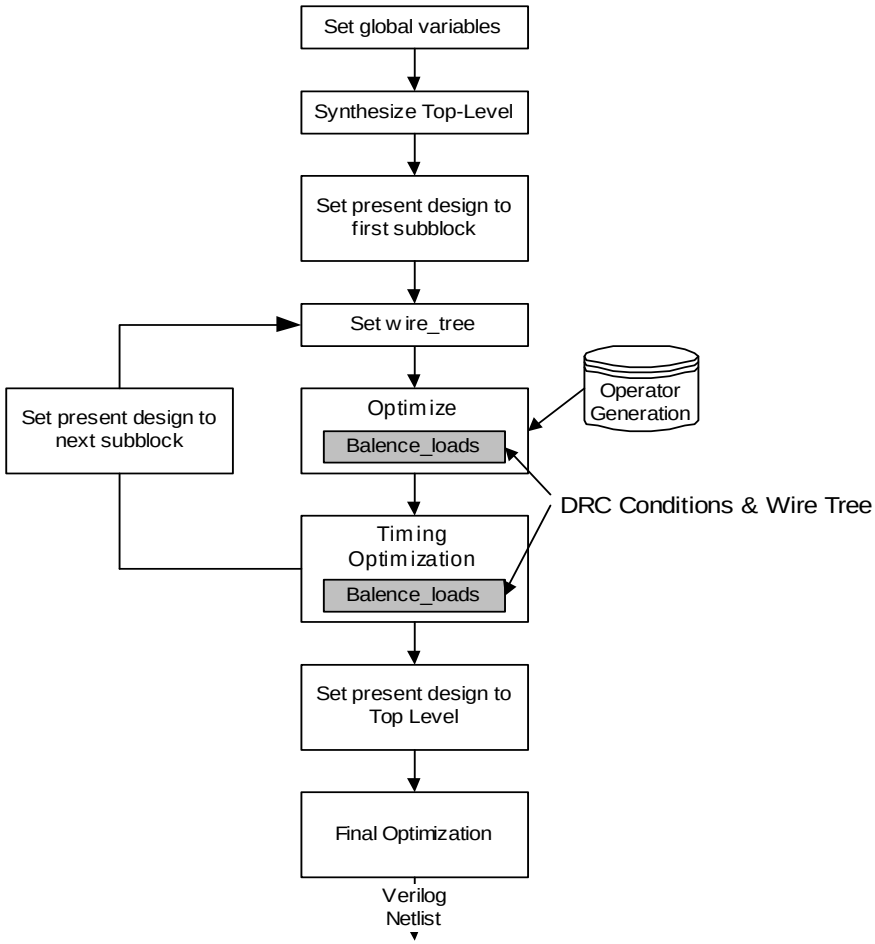
```
group * -inst_name core
optimize –ta cg61 –chip –area –single_level
ungroup core
```

Optimization

A bottom-up optimization methodology is recommended for LeonardoSpectrum.

Performing Optimization

Figure 12-32 Performing Optimization



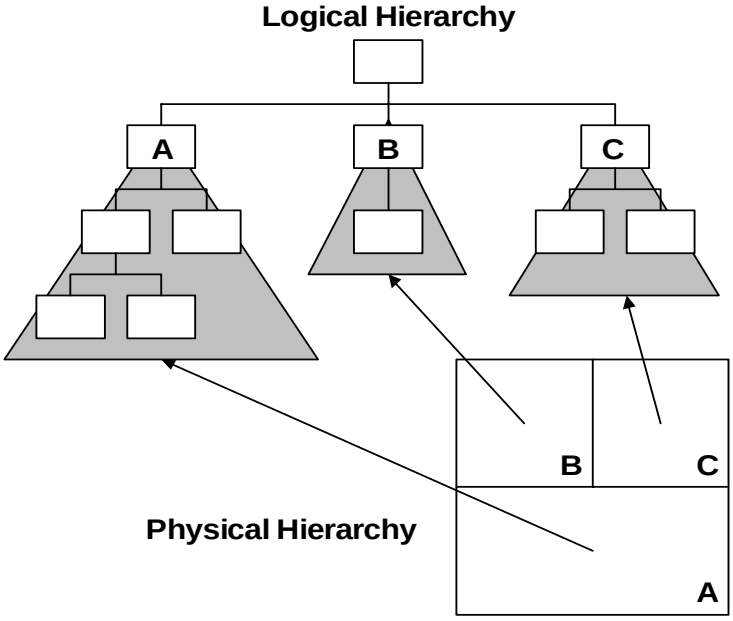
LeonardoSpectrum has been designed to allow easy access to individual sub-blocks for optimization. Therefore, the entire design may be synthesized into generic gates and the `present_design` command used to swap out sub-blocks for optimization. For large designs, you may find that placing one block in memory at a time is convenient.

Hint: LeonardoSpectrum can save the internal database to binary format. After performing synthesis consider saving the **xdb** file for future optimizations runs.

Identify Hierarchy level for optimization

During ASIC floor planning, not every block of chip hierarchy results in physical hierarchy. You must identify the block level that directly corresponds to physical hierarchy and use this as the level for bottom-up optimization. Refer to Figure 12-33.

Figure 12-33 Identify Hierarchy



Optimization Flow

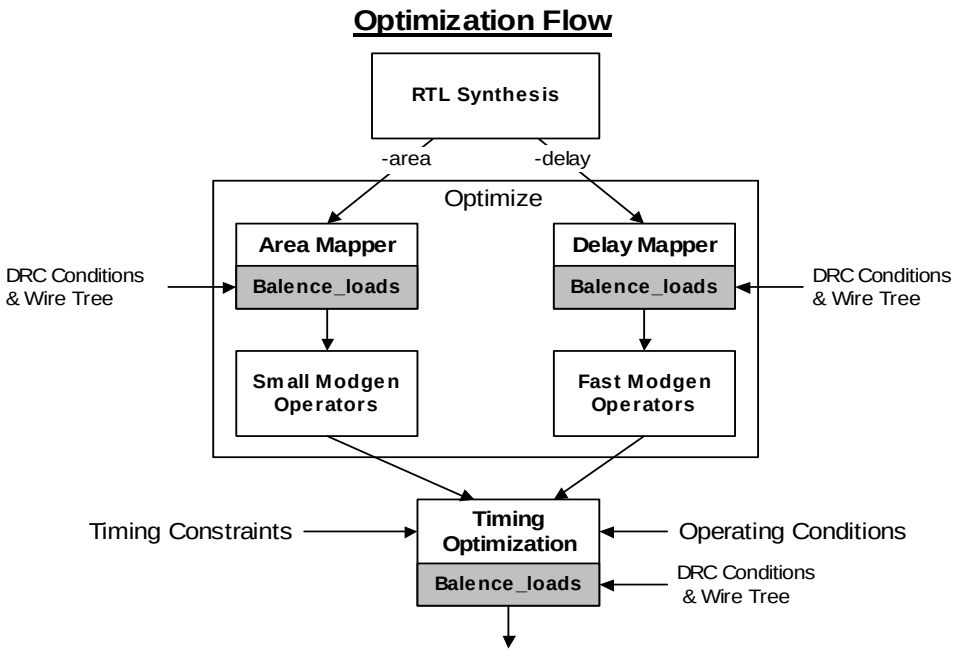
LeonardoSpectrum performs three types of optimizations on an ASIC. Refer to Figure 12-34.

1. **Technology Mapping –** Maps generic gates into either small or fast structures depending on your set switches. Technology mapping also generates either fast or small implementations for all circuit operators. After an operator is placed in the design timing optimization, you cannot modify the operator. Ensure that the correct optimization flag is set during the initial pass.

2. **Timing Optimization –** Performs constraint based, critical path timing optimization on the design.

3. **Balance Loads –** Adjusts circuit fanout and buffering to conform to vendor specific design rules for fanout, capacitance loading, and max transition on a net.

Figure 12-34 Optimization Flow



Optimize:

The `optimize` command performs initial technology mapping and operator generation. If this command is run twice in succession, then the design is unmapped to generic technology. You must remap and re-optimize the design. Results seldom improve with successive optimize runs.

`Optimize` runs the `balance_loads` command automatically. This command corrects design rule violations through logic replication, buffering, and gate sizing. There is no need to run this command independently.

Table 12-5. Arguments to the Optimize Command

Arguments	Description
-target	Specify the target technology for this design
-single_level	Perform optimization only on top level of hierarchy
-effort	Use with optimization -effort levels: remap, standard, exhaustive, and quick
-nopass <list>	Not supported for ASIC
-chip | -macro	-chip inserts i/o buffers for top; -macro does not insert i/o buffers for sub-blocks
-pass <list>	Not supported for ASIC
-area | -delay	Optimize to obtain minimum delay or area (default)

Example:

```
> optimize –ta cg61 –area -macro
```

Timing Optimization

LeonardoSpectrum provides the `optimize_timing` command to improve timing. This command first performs a timing analysis on the design to determine if critical paths are missing timing. If negative slack exists, then `optimize_timing` runs to restructure logic to meet timing.

Table 12-6. Arguments to the Optimize_timing command

Arguments	Description
-through <list>	Specify explicit list of end points to optimize
-single_level	Perform optimization only on top level of hierarchy
-force	Force timing constraints on a block then optimize

```
optimize_timing
```

Balance Loads

The `balance_loads` command is used to correct design rule violations in the circuit through logic replication buffer insertion and gate sizing. Balance loads uses limits defined in the global DRC condition variables to determine when to adjust a circuit. This command is automatically executed during `optimize` and `optimize_timing`. Balance loads must run explicitly on the top-level design with the `-single_level` switch set. This ensures that correct buffering exists between blocks.

Table 12-7. Arguments to the balance_loads command

Arguments	Description
-single_level	Perform optimization only on top level of hierarchy

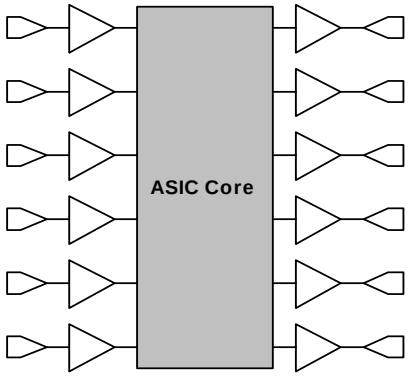
Example:

```
balance_loads
```

IO Buffers

LeonardoSpectrum can automatically insert I/O buffers for ASICs. This is accomplished in two steps. Refer to Figure 12-35.

Figure 12-35 I/O Buffer Insertion



1. Set a pad constraint in the input or output pin

> pad data_in* ITFUH

2. Perform optimization with the “-chip” switch set

optimize –ta cg61 –area –chip

Netlist Unfolding

LeonardoSpectrum, by default, preserves hierarchy in a design. To ensure the fastest possible run times, the netlist is folded which means that all common subblocks reference a single view or netlist. LeonardoSpectrum only optimizes this netlist once.

Note: Refer to the Command Reference guide and to Chapter 4 in this guide for rules on hierarchy auto|preserve|flatten.

If you do two different optimizations on two different instances of common sub-blocks, then the netlist must be unfolded. For example, one block can be optimized for area and a second for delay. Refer to Figure 12-36 and Figure 12-37.

Figure 12-36 Folded Hierarchical Netlist

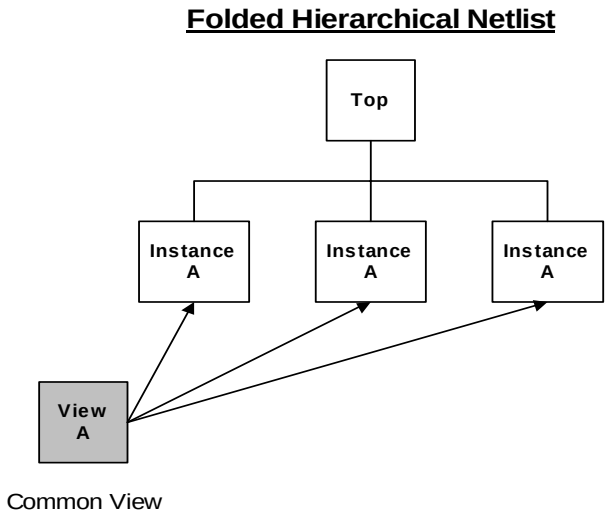
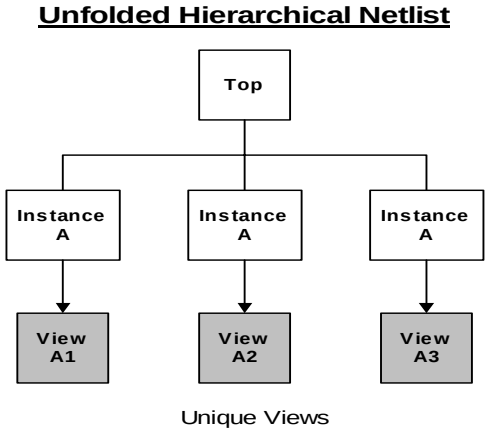


Figure 12-37 Unfolded Hierarchical Netlist



Example Netlist Unfolding Command:

```
> unfold A
```

Final Optimization

Once a design has been stitched together bottom-up, you must generate final area and timing reports. If the design meets specification then one final optimization run is required to insert the GSR reset circuitry and add the chip I/O buffers. To perform final optimizations, run the following command.

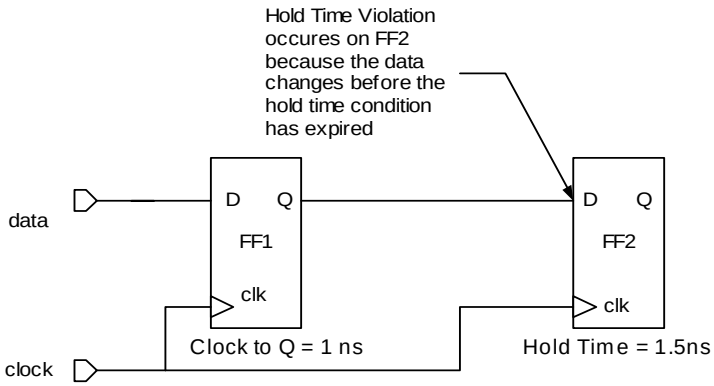
```
> optimize -ta <target technology> -chip -area -no_hierarchy
```

Note: The `-chip` option simply enables the automatic I/O buffer insertion and global reset circuit functions. Otherwise, `-chip` performs the same optimizations as `-macro.`

Hold Time Analysis and Netlist Connection

Although far less frequent than slack time violations, designs can often contain a few hold time violations after optimization has been performed. For this reason a complete hold time analysis should be performed on the design after all slack times have been identified and corrected.

Figure 12-38 Hold Time Analysis



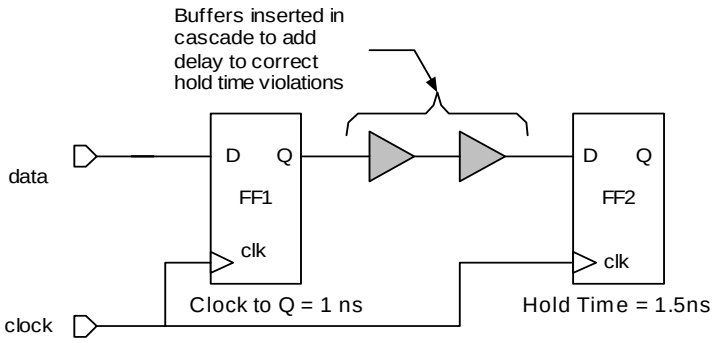
To run a hold time analysis perform the following steps:

1. Set the report_delay_analysis_mode variable to maximum:
   `set report_delay_analysis_mode maximum`
2. Reset timing constraints from minimum values to maximum values, if known.
3. Rerun timing analysis.

Correcting Hold Time Violations

To correct a hold time violation users must manually insert a buffer into the path.

Figure 12-39 Correcting Hold Time Violations



Saving Results

Saving Incremental Results

LeonardoSpectrum supports a native binary database format called XDB. When a design is saved as an XDB file all netlist and timing constraint information is retained. This is the format that is used for intermediate results.

Example:

```
write -format xdb <filename.xdb>
```

Netlisting to place and route

When netlisting to place and route or for gate level simulation use a standard VHDL or Verilog format. For FPGA, the auto_write command performs netlist *tweaking* for backend environments. **Note:** Refer to `auto_write`, Utilities chapter, Command Reference. Use the `write` command directly for ASIC designs. For example,

```
write –format vhdl filename.vhd
```

```
write –format verilog filename.v
```

By default, LeonardoSpectrum converts internal power and ground cells to assign or assert statements in the HDL netlists. This is controlled by the variable `use_assign_for_vcc_gnd`. If you need to set this variable, execute the following command:

```
set use_assign_for_vcc_gnd = TRUE
```

Generating SDF files

LeonardoSpectrum only generates SDF for flat designs. Use these steps:

1. Ungroup all hierarchy
   ```
   ungroup –all –hier
   ```
2. Set the sdf_write_flat_netlist variable to TRUE
   ```
   Set sdf_write_flat_netlist TRUE
   ```
3. Save the SDF file
   ```
   write –format sdf filename.sdf
   ```

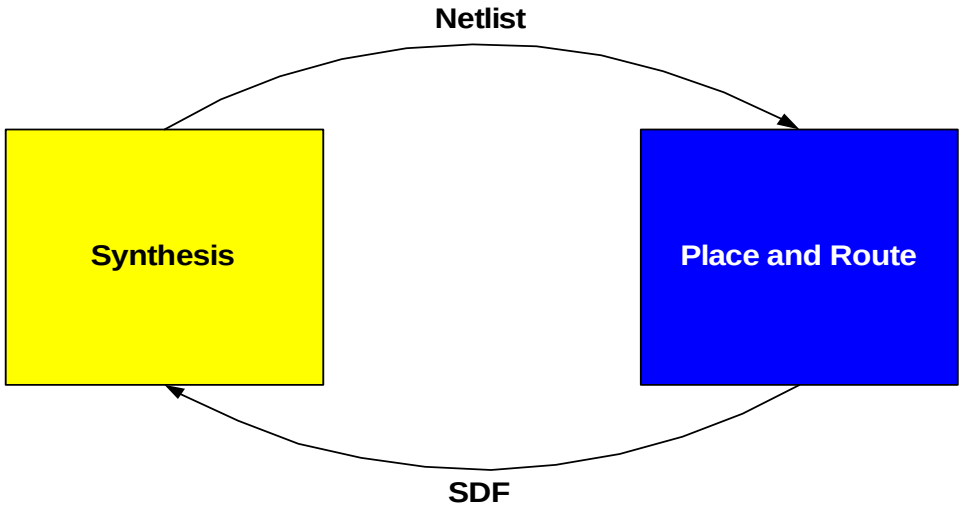
Save and Restore Projects

Refer to Chapter 10.

Back Annotated Static Timing Analysis and Optimization

SDF file can be read into LeonardoSpectrum and incremental optimization is performed using this timing information. Refer to Figure 12-40.

Figure 12-40 Backannotation



Critical Path Schematic Viewing

Place and route tools do not offer schematic viewing. By reading the back annotated design into LeonardoSpectrum, the critical path can be highlighted in the schematic viewer. This provides you with valuable debug information.

Incremental Timing Optimization

By performing static timing analysis, based on the original design constraints, you can verify that additional optimization is not needed. LeonardoSpectrum does incremental timing optimization using this information. When a net or cell is replaced due to logic restructuring, LeonardoSpectrum reverts back to the calculated delays.

Referencing Netlist Objects

Several design manipulation commands reference design objects. The LeonardoSpectrum database is actually quite simple and understanding the elements help you to use these commands effectively. Refer to Chapter 8 for information.

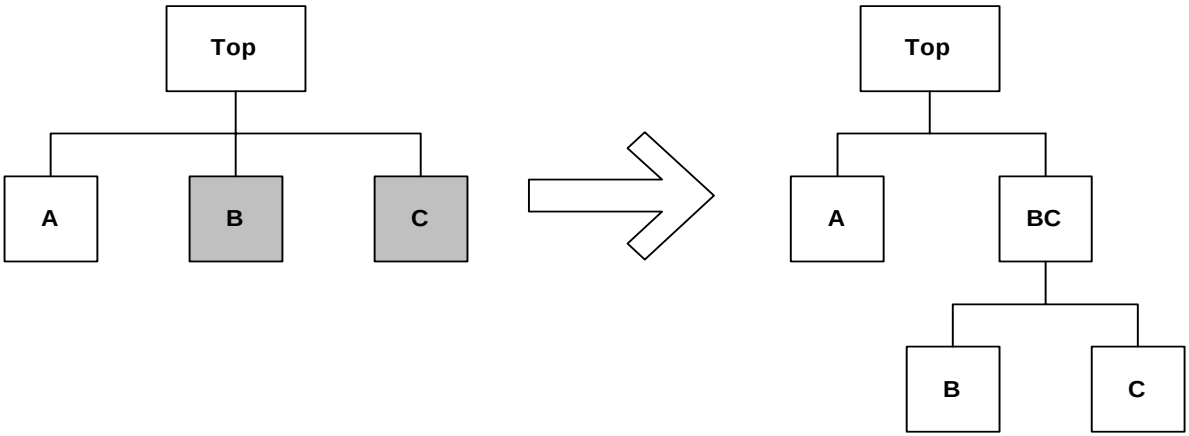
Hierarchy Manipulation

LeonardoSpectrum can perform all common hierarchy manipulations on design instances including `group`, `ungroup`, and `unfold`.

Grouping

Hierarchical grouping refers to the creation of a new hierarchical block from 2 or more selected instances. The original blocks still exist after grouping beneath the new level of hierarchy.

Figure 12-41 Hierarchy Grouping




```
group A B -inst_name AB
```

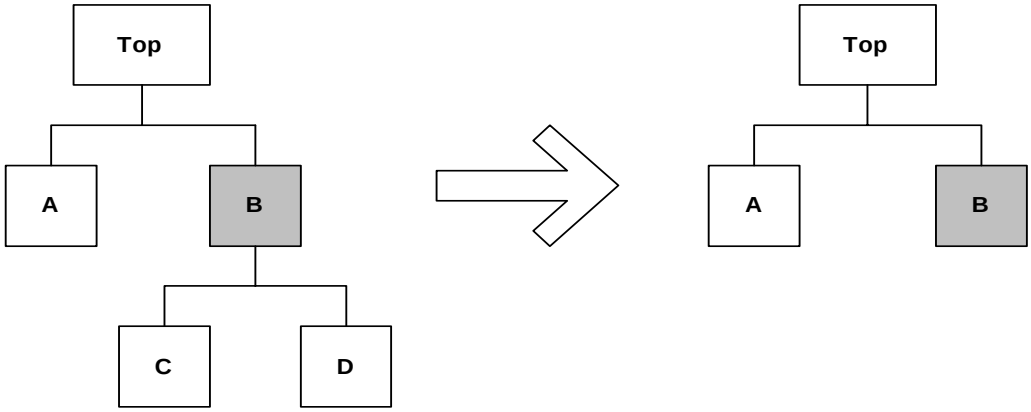
Table 12-9. Arguments to the group command

Arguments	Description
-cell_name	Group a list of instances into one instance of a new view
-view_name	View name of the group
-inst_name	Instance name of the group
-except <list>	Exclude these cells from grouping

Ungroup

Hierarchical ungroup refers to the dissolving of all hierarchy below a selected instance

Figure 12-42 Ungroup



> ungroup B

Table 12-10. Arguments to the ungroup command

Arguments	Description
-all	Ungroups all instances
-hierarchy	Recursively ungroup all hierarchy levels under the selected instance
-simple_names	Use original simple instance names for ungrouped instances
-except <list>	Don't ungroup these instances

LeonardoSpectrum and Design Compiler Commands

```
DC: current_design = diff_in
LS: present_design diff_in

DC: set_operating_conditions "worst3v2" -library "msm13r"
LS: set process min|typical|max

DC: set_wire_load "msm13r2850" -library "msm13r"
LS: set wire_table msm13r2850
(use "report_wire_tree" to view the route tables)

DC: set_dont_touch dlyb
LS: NOOPT dlyb

DC: set_dont_use {msm13r/mx*b, msm13r/mx*e}
LS: set exclude_gates {mx*b mx*e}

DC: compile -map_effort high
LS: optimize -ta msm13r -effort standard

DC: write -format db -hierarchy -o diff_in.db
LS: write -format xdb diff_in.xdb

DC: set_port_is_pad {USB_FSEN, SYS_RST}
pad {USB_FSEN, SYS_RST} IFTUV

DC: create_clock -period 20.8 -waveform {0 10.4} OSC_CLK
LS: Clock_cycle 20.8 OSC_CLK
LS: Pulse_width 10.4 OSC_CLK

DC: all_inputs()
LS: [all_inputs]

DC: all_outputs()
LS: [all_outputs]

DC: set_drive 1 all_inputs();
LS: input_drive 1 [all_inputs]

DC: set_load 1 all_outputs();
LS: output_load 1 [all_outputs]

DC: insert_pads
LS: optimize -ta msm13r -chip -single_level"
(Must set the "-chip" option when running optimize)
```

SynthesisWizard Tutorial 13

Welcome to the SynthesisWizard tutorial. The SynthesisWizard is one of three ways to synthesize your design; Quick Setup and FlowTabs are the other two ways.

The SynthesisWizard consists of four steps that must be completed in the order presented. If you are a first-time user, then the SynthesisWizard is recommended to get you started right away.You can open by clicking on the toolbar SynthesisWizard hat or click Flows->SynthesisWizard. **Note:** While the SynthesisWizard is open, you are restricted entirely to the functions available on the SynthesisWizard.

SynthesisWizard Tour

The following screens and four steps give you a tour of the SynthesisWizard. The steps ask you to apply example choices and to use defaults.

- Step 1 - Technology: Altera FLEX 6k
- Step 2 - Input Files: pseudorandom.vhd (demo file)
- Step 3 - Global Constraints: 20 MHz
- Step 4 - Output File and Finish: pseudorandom.edf (default)

Wizard Buttons

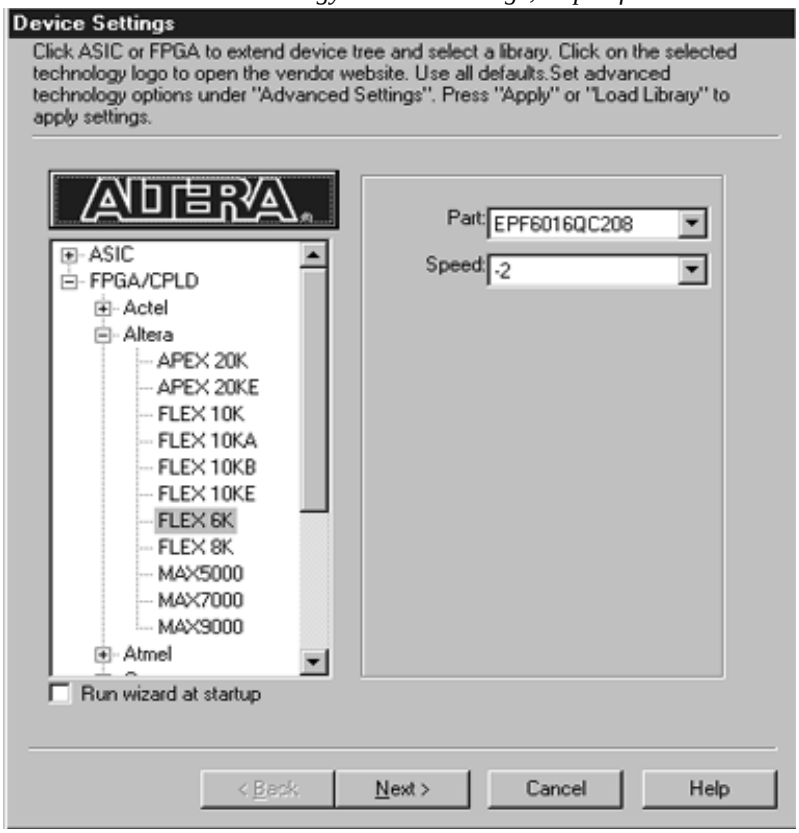
Each of the four SynthesisWizard steps contains buttons that you can click at any time:

- **Help**: select for further assistance.
- **Cancel:** select to exit the SynthesisWizard.
- **<Back**: select to return to the previous SynthesisWizard step, if any.

Device Settings - Step 1 of 4

Use the following steps:

Screen 13-1. FPGA Technology - Device Settings, Step 1of 4

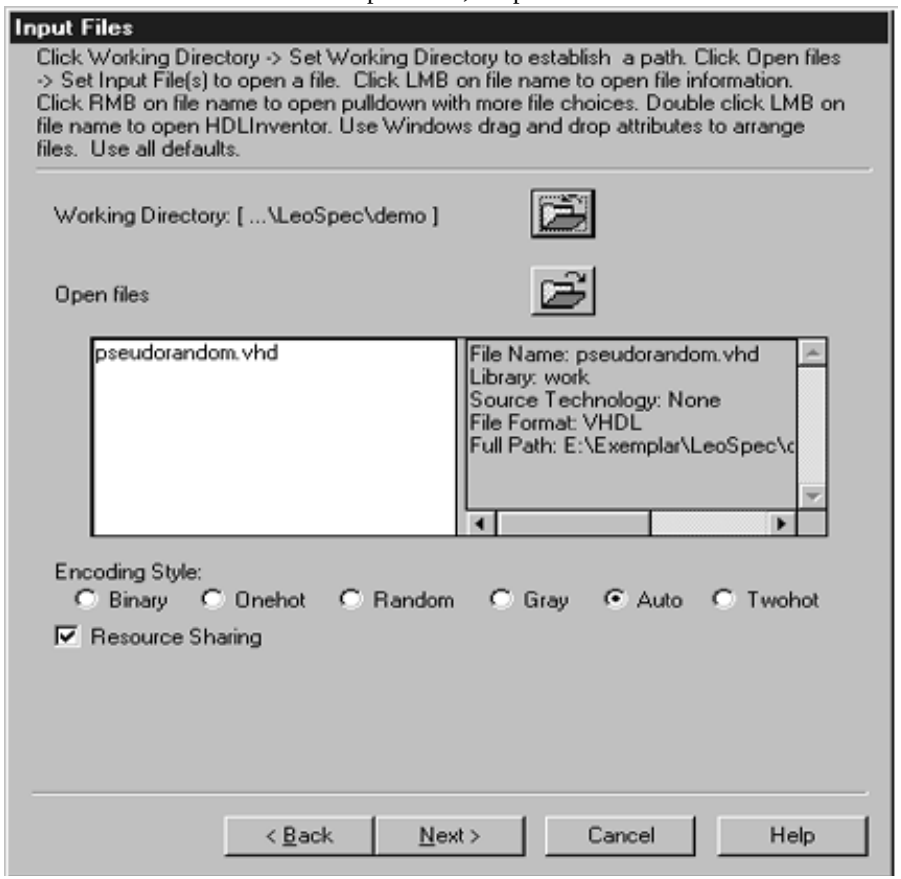


1. Option: Click technology logo to open your default browser and access vendor's Web page, if a Web page is available.
2. Click FPGA to extend the tree and select FLEX 6K.
3. Use defaults for Altera FLEX 6K Part (EPF6016QC208) and Speed (EPF6016QC208-2). If desired change Part or Speed by selecting from pull down.
4. Click **Next>.** Screen 13-2, Input Files opens.

Input Files, Step 2 of 4

Input Files, Screen 13-2, is the second of the SynthesisWizard steps. The input files starts the setup of your HDL design for translation into a gate-level design (netlist) and for place and route. Use the following steps.

Screen 13-2. Input Files, Step 2 of 4

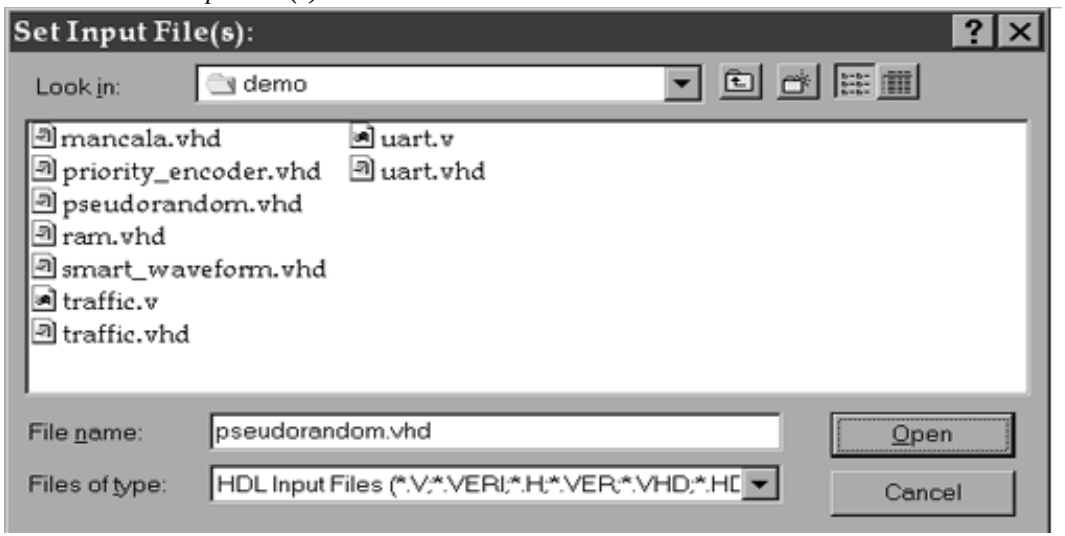

Input Files
Click Working Directory -> Set Working Directory to establish a path. Click Open files -> Set Input File(s) to open a file. Click LMB on file name to open file information. Click RMB on file name to open pulldown with more file choices. Double click LMB on file name to open HDLInventor. Use Windows drag and drop attributes to arrange files. Use all defaults.
Working Directory: [...\LeoSpec\demo]
Open files
pseudorandom.vhd
File Name: pseudorandom.vhd
Library: work
Source Technology: None
File Format: VHDL
Full Path: E:\Exemplar\LeoSpec\c
Encoding Style:
Binary
Onehot
Random
Gray
Auto
Twohot
Resource Sharing
< Back
Next >
Cancel
Help

1. Click on the Working Directory folder to open Screen 13-3, Set Working Directory. Highlight your working directory folder and click Set. Set establishes the path to your working directory which is displayed in the window, and is the default directory for reading and writing files.
   Note: The path to the working directory is also displayed in the status bar near the bottom of main window.
   Note: The working directory is automatically saved and restored between sessions.

Screen 13-3. Set Working Directory

2. Click **Cancel** on Set Working Directory. You now return to Input Files.
3. Next, click Open files button to open Set Input File(s). Refer to Screen 13-4.
4. After you click Open to use pseudorandom.vhd, you return to Input Files. Refer to Chapter 7 for more input file information.
5. Use default Encoding Style. Refer to Chapter 6 and to the HDL Synthesis Guide for binary, gray, onehot, twohot, random, and auto encoding information.

Screen 13-4. Set Input File(s)



6. If desired, change the default ■ Run SynthesisWizard at startup, to prevent the SynthesisWizard from opening at startup.

7. ■ Resource Sharing - If selected allows you to reduce the numbers of certain devices. For example, if your design requires two adders with two inputs each, then you can replace the adders with two muxes and one 2-input adder.

8. Click **Next>.** Screen 13-5, Global opens.

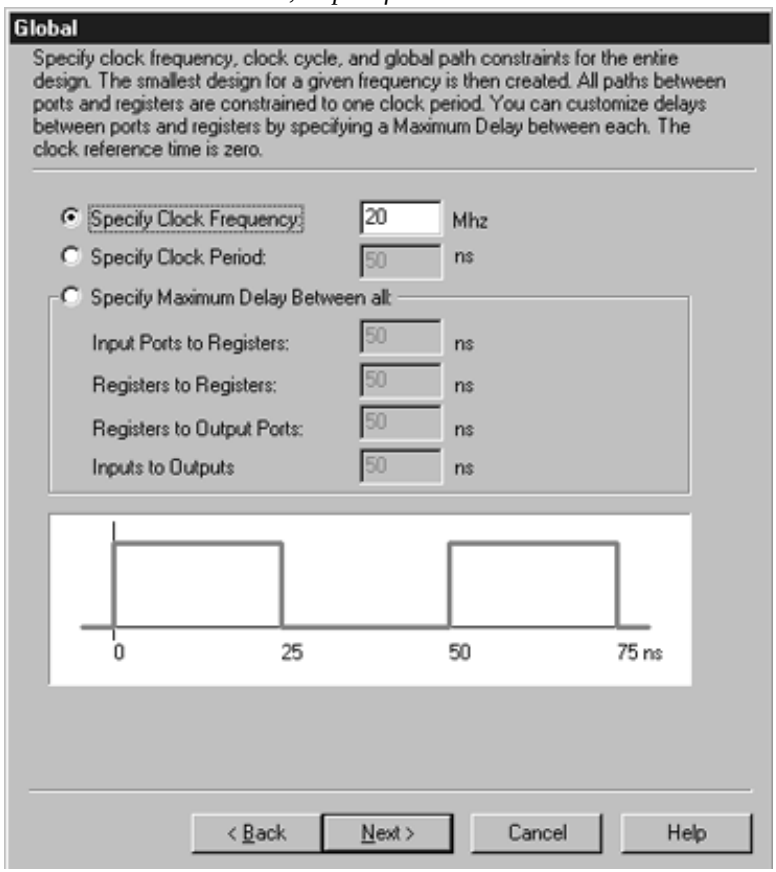
Note: Refer to Chapter 4, Level 3 FlowTabs, Additional Instructions, Retarget an Output Netlist.

Global Constraints, Step 3 of 4

You can set the constraints for the entire design. Use these steps for Screen 13-5:

1. Use global constraint defaults.
2. Type 20 in the Mhz field. A repeating wave form appears in the window with 20 Mhz values.
3. If desired, you can further customize global constraints with the radio buttons.

Screen 13-5. Global Clock, Step 3 of 4

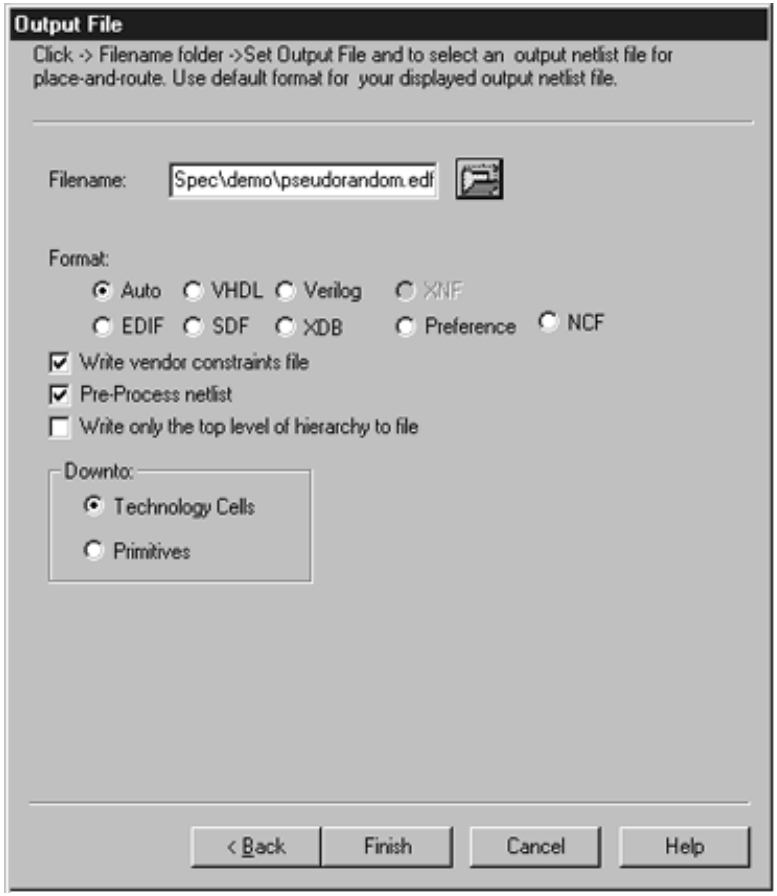


4. If necessary, Refer to Chapter 5 for more constraint information.
5. Click **Next>.** Screen 13-6, Output File opens.

Output File, Step 4 of 4

The Output File, shown in Screen 13-6, allows you to specify the location and format of your FPGA netlist. Refer also to Chapters 3 or 4, Output Tab; and to Chapter 7 for output format information. The Downto: for ●Technology or ❍Primitive Cells includes the selected cells in your output file. Use these steps:

Screen 13-6. Output File, Step 4 of 4



1. Your output file, pseudorandom.edf, is shown in the Filename: field.
2. If desired, click Filename button to change the output file. Screen 13-7, Set Output File, opens.
3. Click **Save/Cancel** to return to Output File, step 4 of 4.

Screen 13-7. Set Output File

4. Use the Format: defaults. Refer to Chapter 7 for information on the output options.

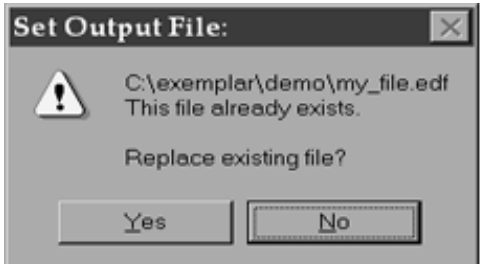
5. Click **Finish**. The wizard closes and the run flow starts.

Run

During run you can view the Transcript in the Information Window and see the entire flow run. The device utilization report for `pseudorandom.vhd` is presented. If you close the information window, click Window -> `pseudorandom.vhd` to open your file again. During synthesizing, the toolbar **Stop** icon turns red to indicate that the system is working. Click **Stop** to stop the run at anytime. The progress of the run appears in the lower left of the status bar. *Ready* indicates that the run is complete; **Stop** is grayed out.

Note: Before the run starts, you are prompted with a warning if an output file already exists. If you click Yes, then the current output file is replaced. Refer to Screen 13-8.

Screen 13-8. Warning - Overwriting Output File



Messages 14

This chapter contains example formats of: Error, Information, and Warning messages.

Information Message

The following information messages may be related to your coding or to the target technology.

Cannot determine the best buffer

Use largest buffer

port %s already contains a PAD cell. IO mapping skipped

not fixing %s DRC violation on net %s because of NOBUFF attribute

fixed %s DRC violation on net %s by resizing driver cell %s to %s

fixed %s DRC violation on net %s by replication

fixed %s DRC violation on net %s by buffering

fixed %s DRC violation on net %s by resizing

Error Message

The following error messages may be related to retargeting a technology or to elaborating.

no %s in library; instance %s cannot be mapped

no %s in library with async set and reset; instance cannot be mapped

transformations variable is FALSE. Cannot transform asynchronous %s of %s

Warning Message

The warning messages may indicate that LeonardoSpectrum is continuing with a flow; however, a black box may be instantiated in your design. LeonardoSpectrum may also apply a contraint, variable, or an attribute by default.

Asynchronous %s in library with both sync and async set and reset; instance %s cannot be found.

no %s in library with both async set and reset; instance %s cannot be mapped.

Index

A

adding a library to the GUI 7-19, 7-20
adding a technology specific symbol library 8-28
Adobe Acrobat Reader 1-8
alphabetical list
  batch mode options 6-1
  command attributes 5-3
Altera
  APEX 4-27
  APEX variables 4-33
  MAX+PLUS II 3-14, 3-18, 4-22, 4-26
  place and route 3-14, 4-22
analysis
  trace to hierarchy 8-5
analyze
  elaborate 4-10, 7-2, 7-3
  read 4-10, 7-3
analyze command 7-2
APEX 20K mapping 4-33
APEX technology
  WYSIWYG primitives 4-33
architecture 7-4
ASIC
  Application Specific Integrated Circuit 1-1
  balance loads 12-59
  clock skew 12-31
  clock uncertainty 12-33
  design methodology 12-1
  design partitioning 12-12
  design rule violations 12-47
  DRC resolving 12-47
  environment setup 12-9
  exclude gates 12-52
  false paths 12-40
  floor planning 12-56
  flow charts 12-2
  flow session 12-5
  global timing 12-29
  I/O buffers 12-60
  instantiate RAM cells 12-22
  library 7-19
  multi-cycle constraints 12-39
  quick setup 4-4
  scaling calculation 12-45
  setting constraints 12-28
  synthesis process 12-15
  technology settings 4-8
  write command 12-65
  XlibCreator 1-9
attribute
  dont_touch 8-26, 8-27
  noopt 8-26, 8-27
  preserve_signal 5-7
attributes 9-2
  commands 5-13

B

back annotation 3-20, 4-28
batch mode 1-4
  command line 6-1
  complex interactive script 1-4
bottom up 7-17
bubble tristates 7-13
buffering specifications, 5-9

C

command
  unfold 8-26
  ungroup -all 8-27
  unmap 8-26, 8-27
constraint
  file 5-1
  set and remove 5-1
constraint editor
  clock 5-18
  global constraints 3-8, 4-12, 5-17
  input 5-20
  module 5-27
  output 5-22
  path constraints 5-29
  report 5-31
  signal 5-24
constraints
  low level block 5-14

conventions
manual 1-11
CPLD
complex programmable logic device 1-1
critical path
instances, nets, and ports 8-9
object query mode 8-9
schematic fragments 8-8
cross probe
file name 8-7
line number 8-7
cross probing 8-2
design browser 8-2
HDL constructs 8-2
HDL source code 8-2

D

database
design 9-1
hdl 7-17
hierarchy 7-17
object 9-2
debug 8-1
descriptions
EDIF input options, 7-7
EDIF output options, 7-8
elaborate input options, 7-2
output files 3-12, 4-20
SDF output options, 7-9
Verilog input options, 7-5
Verilog output options, 7-9
VHDL input options, 7-4, 11-5, 11-7, 11-10
VHDL output options, 7-10
write files 3-12, 4-20
XNF input options, 7-6
design browser 9-2
hierarchy window 8-25
library window 8-24
operators 8-25
popup menus 8-26
primitives 8-8, 8-25
view options 11-8
design names 9-2
absolute 9-3
relative 9-3
DesktopASIC 1-5
dont touch 8-27
DRC
design rule checker 1-5

E

EDIF
input 7-7
output 7-7
exclude gates 7-14

F

Files of type 2-5
filter icons
instance 8-21
net 8-21
pin 8-21
port 8-21
flow
back annotation 3-20, 4-28
synthesis 3-1, 4-1
SynthesisWizard 2-3, 13-1
FlowTabs 3-1, 4-1
FPGA
auto_write 12-65
design methodology 12-1
field programmable gate arrays 1-1
full report 2-12

G

global
specify clock period 3-8, 4-12, 5-17
specify maximum delay 3-8, 4-12, 5-17
group 8-27
GUI
graphical user interface 1-1
GUI options
do not lock LCells 7-14

map to cascades 7-14

H

HDL
hardware description language 1-1
mixing design languages 1-6, 7-17
second language 1-6
HDL constructs
case statement 8-4
declaration 8-3
expressions 8-3
if statements 8-3
instantiation 8-3
procedure call 8-3
variable array indexing 8-4
HDL solution 1-1
HDLInventor
bookmarks 2-16
cross probing 8-2
templates 2-14
help
about 2-10
contents 2-10
online 1-9
Hierarchical
grouping 12-67
hierarchical
ungroup 12-68
hierarchical grouping 12-67
hierarchy
auto-dissolve 3-2, 4-2, 4-4, 4-14
flatten 3-2, 4-2, 4-4, 4-14
preserve 3-2, 4-2, 4-4, 4-14
hierarchy jump symbols 8-8, 8-10

I

icon
editing 2-11
Exemplar's logo 2-11
schematic 8-16
task 2-11
information window
filtered transcript 2-13
transcript 2-16
input drive characteristics, 5-6
input/output timing parameters, 5-11
interactive command line shell
design database 9-1
list commands 9-6
list variables 9-6
script 2-12
invalid schematic 8-2

J

jump symbols 8-8, 8-10

L

Lattice/Vantis
place and route 3-18, 4-26
LeonardoInsight
critical path 8-10
cross probing 8-2
design browser 1-5
schematic fragments 8-8
schematic viewer 1-6
LeonardoSpectrum user's manual 1-8
Level 1 features 1-2
Level 2 features 1-3
Level 2 tab
global constraints 3-8
input 3-6
output 3-12
P&R 3-14
quick setup 3-2
technology 3-4
Level 3
interactive command line shell 1-11, 2-12, 5-1, 9-2
Level 3 features 1-4
Level 3 tab
global constraints 4-12
input 4-10

optimize 4-14
optimize timing 3-20, 4-28, 7-17
P&R 4-22
quick setup 4-2, 4-4
quick setup ASIC 4-4
report 4-16
technology 4-6
technology ASIC 4-8
libraries
ASIC technologies 7-19
FPGA technologies 7-19, 7-20
symbol, 8-28
synthesis, 1-9
library 9-1
cell 9-1
work 9-2
line numbering 2-14
list_attributes command 9-3
list_design command 9-3
low level blocks 7-17

M

main window
banner 1-11
information window 2-12
menu bar 2-5
status bar 2-19
tool bar 2-11
menu bar
analysis 8-5
messages
error 14-1
information message 14-1
warning 14-2
mixing design languages 7-17
Model Sim/QuickHDL simulator 1-4
multicycle path 5-13, 5-29

N

neoprim netlist 1-7
net 8-8, 8-9, 8-10, 8-11, 8-13, 8-18, 9-2

O

object names 9-4
objects 9-2
online, 1-8
operators 9-1
optimization
-effort ASIC option 4-4
option
HDL languages 1-6
LeonardoInsight 1-5
options
-architecture=<name>, 6-1
-area, 6-2
-auto_fast_io, 6-23
-auto_register_packing, 6-23
-batchhelp 6-2
-bus_name_style, 6-2
-chip, 6-2
-command_file=<file_name>, 6-2
-control=<file_name>, 6-3
-crit_path_analysis_mode=both, 6-3
-crit_path_analysis_mode=maximum, 6-3
-crit_path_analysis_mode=minimum, 6-3
-crit_path_arrival=<float_number>, 6-3
-crit_path_detail=short|full, 6-3
-crit_path_from=<node_list>, 6-3
-crit_path_longest, 6-4
-crit_path_no_int_terminals, 6-4
-crit_path_no_io_terminals, 6-4
-crit_path_report_input_pins, 6-4
-crit_path_report_nets, 6-4
-crit_path_rpt=<file_name>, 6-4
-crit_path_slack=<float_number>, 6-5
-crit_path_to=<node_list>, 6-3
-crit_paths_not_thru=<node_name>, 6-5
-crit_paths_thru=<node_name>, 6-5
-delay 6-2
-design=<name>, 6-5
-dont_lock_lcells, 6-5
-edif=<file_name>, 6-5
-edif_file=<file_name>, 6-5
-edif_timing_file 6-6
-edifin_ground_net_names 6-6
-edifin_ground_port_names 6-6

-edifin_ignore_port_names 6-6
-edifin_power_net_names 6-6
-edifin_power_port_names 6-6
-edifout_ground_net_name 6-6
-edifout_power_ground_style_is_net 6-6
-effort=<effort_level>, 6-7
-enable_dff_map_optimize 6-7
-encoding=<encoding_style>, 6-8
-entity=<name>, 6-9
-exclude=<gate_name>, 6-9
-file=<script name> Level 3, 6-9
-full_case, 6-9
-generic=<name> <value>, 6-9
-global_sr=<name>, 6-9
-help, 6-10
-hierarchy_auto, 6-10
-hierarchy_flatten, 6-10
-hierarchy_preserve, 6-10
-highlight_file 6-10
-include=<gate_name>, 6-10
-input_format=<file_type>, 6-10
-insert_global_bufs 6-10
-lock_lcells, 6-5
-log_file 6-11
-lut_max_fanout 6-11
-macro 6-2
-map_area_weight=<integer>, 6-11
-map_delay_weight=<integer>, 6-11
-map_muxf5 6-11
-map_muxf6 6-11
-max_cap_load=<integer>, 6-11
-max_exe, 6-23
-max_fanin=<integer>, 6-12
-max_fanout_load=<integer>, 6-11
-max_frequency=<integer>, 6-12
-max_pt=<integer>, 6-12
-max_ta_reg, 6-23
-maxarea=<integer>, 6-11
-maxdly=<integer>, 6-12
-maxplus2, 6-23
-modgen_library=<name>, 6-12
-module=<name>, 6-12
-ncf_filename 6-12
-no_acf, 6-23
-nobreak_loops_in_delay, 6-13
-nobus 6-13
-nocascades, 6-13
-nocheck_complex_ios, 6-13
-nocomplex_ios, 6-13
-nocontrol, 6-3
-nocounter_extract, 6-13
-nocrit_path_rpt, 6-4
-nodecoder_extract, 6-13
-noenable_dff_map 6-7
-noglobal_symbol, 6-13
-noinfer_global_sr, 6-14
-nologic_rep, 6-14
-nolut_map, 6-14
-nomap_global_bufs, 6-14
-noopt=<gate_name>, 6-14
-nopack_clbs, 6-14
-nopass=<pass_number>, 6-17
-nopld_xor_decomp, 6-15
-noram_extract, 6-15
-nosdf_hierarchical_names, 6-15
-nosummary, 6-19
-notimespec_generate, 6-15
-notransformations, 6-15
-nowire_table, 6-22
-nowrite_lut_binding, 6-15
-noxlx_fast_slew, 6-16
-noxlx_preserve_pins, 6-16
-num_crit_paths=<number_paths>, 6-16
-optimize_cpu_limit=<seconds>, 6-16
-output_format=<file_type>, 6-16
-package=<name>, 6-17
-pal_device 6-17
-parallel_case, 6-17
-part=<part_number>, 6-17
-pass=<pass_number>, 6-17
-preference_filename=<file_name>, 6-17
-preserve_dangling_net, 6-17
-process=<name>, 6-18
-product=<ls1| ls2| ls3>, 6-18
-propagate_clock_delay, 6-18
-report_brief, 6-18
-report_full, 6-18
-sdf_hier_separator=<separator_character>, 6-18
-sdf_in=<file_name>, 6-18
-sdf_names_style=vhdl|verilog|none, 6-18
-sdf_out=<file_name>, 6-18

-sdf_type=min|typ|max, 6-19
-select_modgen=fast|fastest, 6-19
-select_modgen=smallest|small, 6-19
-session_file 6-19
-simple_port_names, 6-19
-source=<library_name>, 6-19
-summary=<file_name>, 6-19
-target=<library_name>, 6-20
-temp=<degrees>, 6-20
-tristate_map, 6-20
-use_f5map, 6-20
-use_f6lut, 6-20
-use_qclk_bufs, 6-20
-verilog_file 6-20
-verilog_wrapper=<wrapper_file>, 6-20
-vhdl_87, 6-21
-vhdl_93, 6-21
-vhdl_file 6-21
-vhdl_wrapper=<wrapper_file>, 6-21
-vhdl_write_87, 6-21
-vhdl_write_bit=<type>, 6-21
-viewlogic_vhdl, 6-21
-voltage=<volts>, 6-21
-wire_table=<name>, 6-22
-wire_tree=best|balanced|worst, 6-22
-write_clb_packing, 6-22
-xlx_preserve_gsr, 6-22
-xlx_preserve_gts, 6-22
output bus 8-14
output load characteristics, 5-6

P

P&RIntegrator 1-2, 1-7, 3-20, 4-28
PackagedPower 1-12
LeonardoSpectrum 1-12
ModelSim 1-12
Renoir 1-12
parameters 7-3
PDF manual 1-8
place and route
Altera 3-14, 3-18, 4-22, 4-26
Altera Quartus 3-19, 4-27
Xilinx 3-16, 4-24
pop_design script 9-3
popups
context-sensitive 8-9
port instance 9-2
power tab
advanced technology 7-14, 7-16
EDIF input 7-7
EDIF output 7-8
elaborate 7-2
SDF output 7-9
Verilog input 7-5
VHDL input 7-4
VHDL output 7-10
XNF input 7-6
power tabs 3-1, 4-1
PowerTabs
EDIF 7-8
elaborate 7-2
input 7-4
SDF 7-9
Verilog 7-9
VHDL 7-10
present_design command 9-3
preserving signals 5-7
primitives 9-1
project
input files 10-9
interactive shell 10-9
large design 10-9
new 10-8
open 2-7
optimization 10-7
platform to platform 10-10
portable 10-8
save 2-7, 10-9
save and restore 1-6
save as 2-7
synthesis runs 10-8
tips and tricks 10-9
tips for using 10-10
project files
.lsp file 2-6, 10-7
.src file 10-7

.xdb file 10-7
project Level 3
checkpoint 10-7
projects
recent 2-7
push_design script 9-3

Q

Quartus
place and route 3-19, 4-27
Quick Setup
Options 4-2
retarget 4-2
quick setup
ASIC 4-4
FPGA 3-2, 3-3, 4-2

R

ready message 2-19, 13-8
register timing, 5-11
Renoir graphical design 1-12
report
all leafs 4-16
cell usage 4-16
levels of hierarchy 4-16
Retarget 4-30
retarget
output netlist 3-6, 4-2, 4-10, 4-30
right mouse button (RMB), 1-2, 8-4
RMB 2-3, 2-16, 3-6, 8-9, 8-11, 8-17, 8-22, 8-24, 8-26, 8-27
root 9-2, 9-3
libraries 9-1
RTL
schematic 8-8
templates 2-14
run
warning prompt 13-8

S

schematic
gate level 8-8
icons 8-16
invalid 8-2
partition 8-16, 11-7
RTL 7-13, 8-8
search 8-19
strokes 8-23
schematic cones
fanin 8-11
fanout 8-11
scripts 6-1, 9-5
SDF for flat designs 12-65
session settings
editor options 11-2
options 11-4
schematic viewer properties 11-6
shortcuts popup 3-6, 4-10
simprim netlist 1-7
standard features
design browser 1-8
HDLInventor 1-7
P&RIntegrator 1-6
save and restore project 1-6
status bar
flow progress messages 2-19
line counter 2-19
ready message 2-19
toolbar messages 2-19
working directory path 2-19, 13-3
syntax
buss specification, 5-2
constraint file commands, 5-2
SynthesisWizard
FPGA technology 13-2
global clock 13-6
input files 13-3
output file 13-7
retarget 13-5
step 1 of 4, 13-2
step 2 of 4, 13-3
step 3 of 4, 13-6

step 4 of 4, 13-7

T

Tcl 6-1, 6-9, 9-5
Tcl script
level 2 2-6
level 3 2-6
three ways to synthesis
FlowTabs 2-3, 13-1
quick setup 2-3, 13-1
SynthesisWizard 2-3
timing requirements, 5-10
tip of the day 2-2
toolbar
editing icons 2-11
stop icon 13-8
tools
schematic viewer 8-1
variable editor 11-9
top entity 7-4

U

unfold 8-25
ungroup 8-16, 8-27
ungroup all 8-27
unmap 8-25, 8-27

V

variable editor 11-10
variables
dont_lock_lcells 7-14
flex_use_cascades 7-14
Verilog
full case 7-5
input 7-5
parallel case 7-5
top module 7-5
version number 2-10
VHDL
architecture 7-4
attributes 5-13
video demo 2-10
view 9-1
instance 9-2
port 9-2

W

wave form 13-6
web site 1-9
wildcards 9-3
wizard
step 1 of 5, 13-3
wizard screens
select FPGA technology 13-2
select input file(s), 5-18, 5-20, 5-22, 5-24, 5-26, 5-28, 5-30
working directory 13-3

X

XDB database format 12-65
XDB format 4-6, 4-20
Xilinx
M1 3-16, 4-24
place and route 3-16, 4-24
Xilinx library
Neoprim 4-33
Simprim 4-33
XlibCreator 1-9
XNF
preserve dangling nets 7-6
preserve GSR signal 7-6
preserve GTS signal 7-6